Quick BASIC テクニカルマニュアル

オペレーションと構造化プログラミングの基本

伊東ひろみ

RUN

EDIT

DEBUG

Technical Manual

Quick BASIC

テクニカルマニュアル

オペレーションと構造化プログラミングの基本

伊東ひろみ

新星出版社

はしがき

アプリケーションソフトの時代？・・・

パソコンの世界は「秒進分歩」。より新しく高性能なハードウェア、より楽しく実用的なソフトウェアが次から次へと送り出されています。

つい何年か前には、パソコンとBASICは切っても切れないものでした。パソコンを利用するためには、自分でBASICを使ってプログラムをつくらなければならなかったのです。

いまではそんなことはありません。便利で使いやすいアプリケーションソフトがたくさん発売されています。これからパソコンとお付き合いを始めるみなさんは、しごとや趣味にぴったり合ったパソコンと市販のアプリケーションソフトを買ってきて、利用すればそれで十分なのです。

ところが、Quick BASICが発売されて以来、これがプログラミング言語ソフトにもかかわらず、その人気は一般のアプリケーションソフトを越えてしまうほどの予想外の販売数を記録しています。

すでにコンピュータのプログラミング言語を学んで、自分用のソフトをつくってパソコンを利用する時代は終わっていたはずなのです。

現在では、多大の資本を投資して何十人ものプロフェッショナルが、１年も２年もかけてアプリケーションソフトを開発しています。もちろん、これに匹敵するようなプログラムを個人でつくるのは困難です。ましてや自分のしごとや家業に使用するソフトを自分でつくるなどということは考えられない時代です。では、なぜ「いま、プログラミング言語」なのでしょう。

Quick BASICの世界へようこそ・・・

パソコンはしごとや趣味の道具というだけではなく、パソコンに触れることそのものが楽しみや趣味の対象となり得るものです。

自分でいろいろなプログラムをつくることは、絵を描いたり模型をつくったり、テレカやＣＤを収集したり、音楽をつくったり歌ったりすることと同じように楽しいことなのではないでしょうか。

Quick BASICは、パソコンでプログラムをつくるためのたいせつな道具だてを十分に用意して、使いやすい環境を整えています。Quick BASICの環境

は、パソコンの初心者にもベテランにも使いやすく、プログラムを効率的に作成できるように考えられています。

このような優れた環境がQuick BASICの人気の秘密でしょう。

けれど、プログラミング言語を覚えるのは、なんといってもパソコンを自由に使いこなすという楽しみのためではないでしょうか。Do It Yourselfの精神で、自分の使うプログラムを自分でつくり上げるのが楽しいのです。

パソコンは、ちょっとでもまちがったプログラムは受け付けてくれない、石頭の機械です。この石頭くんを思いどおり使いこなすのはたいへんですが、苦労してつくったプログラムが動いたときには、やはり快感です。

Quick BASICを越えて・・・

Quick BASICを使いこなすことは、パソコンの操作や利用法を学ぶことでもあります。

Quick BASICは、キーボードやマウスを使って操作します。これは他のいろいろなアプリケーションソフトでも同じことです。

パソコンは、ソフトによって種々の働きをする汎用機です。

Quick BASICの環境を使いこなすことによって、パソコンの基本的な操作法や、利用のノウハウを覚えられ、他のアプリケーションソフトを操作するときにも類推して早く使いこなせるようになります。

また、BASICのプログラムを作成することによって、パソコンの働きや、ソフトのしくみなども理解できるようになります。

こうしてハード、ソフトの知識が増えると、他のパソコンやいろいろなアプリケーションソフトなども臆せず利用できるようになることでしょう。

本書では、Quick BASICの環境を使いこなす方法と、BASICの基本ステートメントを使ってのプログラミングの方法を解説しています。

本書が、Quick BASICを通してあなたのパソコンライフを広げてくれる一助になれば幸いです。

伊東ひろみ

もくじ

PART 0
はじめましてMS-DOS

PART 1
レッツQuick BASIC

PART 2

画面と応用いろいろ

PART 3
Quick BASICでなにかする

PART 4
プログラミングの入口

PART 5
Quick BASICプログラマー

PART 0

はじめまして MS-DOS

Quick BASIC と MS-DOS

MS-DOS 上で動く Quick BASIC

「Quick BASIC は MS-DOS の上で動きます」

こうはいっても、Quick BASIC をなにかの上に乗せて動かすわけではありません。「MS-DOS」というのは、パソコンで言語やワープロソフトなどのプログラムを働かせるための「基本ソフト」のひとつです。

MS-DOS は「Microsoft-Disk Operating System」の略。アメリカのマイクロソフト（Microsoft）社が16ビットパソコン用に開発した OS です。

OS と DOS のしごと

「OS」は、コンピュータを動かすための基本的なソフトウェアの集まりのことで、「DOS」は、フロッピーディスクやハードディスクへのファイルの読み書きの機能を追加したものです。

この OS を使わなければ、周辺機器の厳密な制御は、すべてコンピュータのプログラムをつくる人が行わなければなりません。周辺機器の機械的な構造や電気的なしくみをすべてわかってからでないと操作できないことになります。

普通の人にとってパソコンはもちろん、フロッピーディスクへの読み書き、ディスプレイへの表示、キーボードからの文字入力などの操作をひとつひとつ細かく制御するなどということは、現実的に不可能です。

それにもまして、ソフトごとに違うコンピュータや周辺機器を特定のものに定めて、プログラムをつくらなければならなくなってしまいます。

フロッピーディスクにデータを記録する方法も、プログラムをつくった人それぞれがすべて違っていたのでは、プログラムごとにすべてのデータを入れ直さなければなりません。

プログラムや機械ごとに違っているその使い方を統一しておけば、どんなソフトでも、同じ方法で周辺装置を取り扱えます。データも同じ形式で記録しておけば、複数のソフトでデータを共通に利用することができます。

OS は、このようなプログラムや、機械ごとにバラバラの仕様を統一して、操作できるようにするしごとをしています。

OS
MS-DOS

DOS は OS のしごとのほか、フロッピーディスクやハードディスクなどの操作を通じて、特にデータの統一性を確保するしごとをしています。

MS-DOS の直接操作も必要になるから

Quick BASIC は、MS-DOS を仲立ちにして、パソコン本体とデータの入力装置であるキーボード、出力装置であるディスプレイ、プリンタ、そしてデータの記録装置であるフロッピーディスク、ハードディスクを操作しています。

しかし、Quick BASIC には基本ソフトである MS-DOS は組み込まれていません。

Quick BASIC を使い込んでいくうちに、ファイルの操作や他の言語でつくったプログラムとのリンクといった複雑な操作が必要になってきます。そうなってくると、どうしても MS-DOS を直接操作しなければならなくなります。

そこで、ここでは MS-DOS の基本的なコマンドの機能と操作法を簡単に説明しておきます。

MS-DOS を起動する

MS-DOS のシステム

MS-DOS のシステムは、

IO. SYS
MSDOS. SYS
COMMAND. COM

の 3 つのファイルからできています。

このうち COMMAND. COM 以外のファイルは DIR コマンド（後述）で画面に表示させて見ることはできません。

IO. SYS は、MS-DOS がハードウェアを直接制御して、パソコンとソフトの仲立ちをするプログラムです。

MSDOS. SYS は、おもにフロッピーディスクやハードディスクなどのファイルの管理を行い、データの共通化に役だっています。

COMMAND. COM は、キーボードから入力されるコマンド（命令）を解釈して実行します。

内部コマンドと外部コマンド

コマンドのうち、初めから COMMAND. COM に組み込まれているものを「**内部コマンド**」、フロッピーディスクやハードディスクに登録されていて、指定されるたびにパソコン本体のメモリに読み出されて実行されるものを「**外部コマンド**」と呼びます。

内部コマンドは、頻繁に利用されるコマンドが標準で組み込まれています。

外部コマンドは、処理の複雑なものや、頻繁にバージョンアップ（機能強化）の行われるものが標準で登録されています。

内部コマンド、外部コマンドは意識して使い分ける必要はありませんが、指定した外部コマンドが指定したドライブやディレクトリにない場合にはエラーになってしまいます。

IO. SYS　CONFIG. SYS　DEVICE
MSDOS. SYS　BUFFERS　AUTOEXEC. BAT
COMMAND. COM　FILES　バッチファイル

MS-DOSの起動

MS-DOSは、以上の3つのファイルがあれば起動できます。この3つのファイルの入ったフロッピーディスクを「**システムディスク**」と呼びます。

システムディスクをA：ドライブに入れて、本体のスイッチをONにすれば、MS-DOSが起動します。このときに、

CONFIG. SYS
AUTOEXEC. BAT

の2つのファイルを使います。

CONFIG. SYSは、MS-DOSが起動するときに、システムの構成を指定するために使われるファイルです。おもに次の構成を設定します。

BUFFERS　フロッピーディスクやハードディスクからデータなどを読み出すときのバッファ（データを一時蓄えておく場所）の量を指定します。多くとるとディスクへのアクセスが早くなりますが、メモリを多く使ってしまいます。

FILES　一度に取り扱うことのできる最大のファイル数を指定します。

DEVICE　日本語FEPやマウス、プリンタなどのデバイス（周辺機器、ソフト）を設定します。

標準的な設定例の画面は、次ページに掲載します。

AUTOEXEC. BATは、自動実行ファイルと呼ばれ、MS-DOSが起動したときに、いちばん最初に実行されるバッチファイル（後述）です。

このファイルのなかに、実行するコマンドを設定しておけば、自分の利用しやすい環境に設定したり、アプリケーションソフトのオートスタートを行うことができます。

■初期設定の画面例

```
A>TYPE CONFIG.SYS
SHELL=A:¥COMMAND.COM A:¥ /P
FILES=20
BUFFERS=10
device=atok6a.sys /d=b:¥atok.dic /s=1 /e=1 /t=1 /b=0
device=atok6b.sys

A>
```

（設定データ）
BUFFERS：
FILES：
DEVICE：

ドライブ／ディレクトリ／ファイル

ドライブは情報の集まった図書館の書架

私たちはおもに、フロッピーディスクやハードディスクのファイル操作を行うために、MS-DOS を利用しています。

MS-DOS では、これらのフロッピーディスクやハードディスクに名まえをつけてそれぞれの機器を指定しています。

ドライブは、そのディスクに入っている情報の総体を示すものです。ある図書館に集まって整理されている書籍のうち、例えば「文学」という大きな分野をまとめて収納した書架のようなものと考えてください。

ドライブ名は、A：、B：、C：のようにアルファベットと「：」（コロン）を組み合わせています。

PC-9801 シリーズの場合は、ドライブ1がAドライブに、ドライブ2がBドライブに対応しています。ハードディスクやRAMディスクを使っている場合は、これとは違う対応になっている場合もあります。

ディレクトリは分類された情報の各分野

ドライブが図書館の書架だとすると、ディレクトリは、書架を分類に従っていくつかに分けて整理した部分ということができます。

例えば、Ａドライブに入っているフロッピーディスクには「文学」についてのデータが入っていたとしましょう。

このフロッピーディスクのなかに「日本文学」、「外国文学」などに分けていくつかの文学のデータが入っています。ディレクトリとはこの「文学」や「日本文学」、「外国文学」の分類のことを表わしています。

ディレクトリには、適切な「名まえ」をつけて管理します。名まえは、アルファベットやカナ文字（１バイト文字という）など８文字までです。

●カレントドライブとカレントディレクトリ

現在、操作をするために選択されているフロッピーディスクやハードディスクのドライブを「**カレントドライブ**」といいます。

カレントドライブを変更するには、キーボードから直接新しいドライブ名を入力します。

例えば、カレントドライブがＡで、プロンプトが「Ａ＞」となっている場合、

B：⏎

とすると、カレントドライブがＢになり、プロンプトもカレントドライブを表わす「Ｂ＞」になります。

同じように、現在操作をするために選択されているフロッピーディスクやハードディスクのドライブ中のディレクトリを「**カレントディレクトリ**」といいます。

カレントディレクトリを変更するには、MS-DOSのコマンド「CD」を使います（後述）。

●階層ディレクトリ

ディレクトリは、図のようにちょうど木の枝を逆さにしたように、ルート(根)ディレクトリから、先（下）へ先へと伸ばしていくことができます。

ルートから見ると、ピラミッド型の構造のなかをだんだんと階層が下がっていくように見えるので、「階層ディレクトリ」といいます。

先ほどの図書館の例でいえば、書架がドライブ、ルートディレクトリが「文学」、その下の階層ディレクトリが「日本文学」や「外国文学」になります。

さらにその下の階層をつくるとすると、「日本文学」の下には「古典文学」や「現代文学」、またその下では「小説」「戯曲」……などに分類されて、ディレクトリを深くしていくことになります。

B：¥BUNGAKU¥NIHON¥KOTEN

というディレクトリが構築されている場合、Bドライブのフロッピーディスクのいちばん上の階層が、つまり「BUNGAKU（文学)」が「ルートディレクトリ」です。これは、「¥」(円マーク）で表わされています。

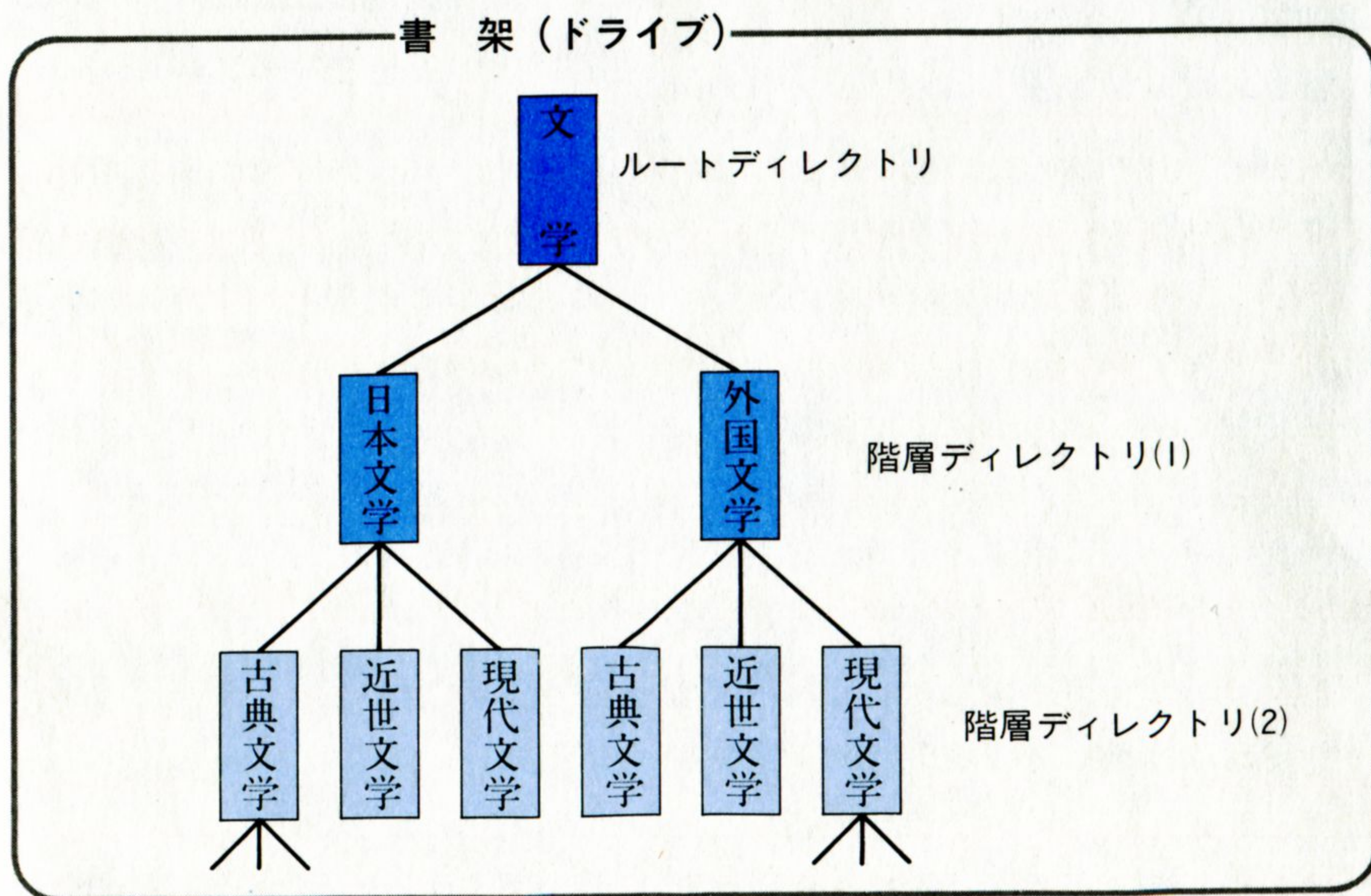
次の階層が、「NIHON（日本)」です。

その下が「KOTEN（古典)」で、上のディレクトリとの区切りはやはり「¥」です。さらにその下の階層があれば、「SHOUSETU」などとつづきます。

書　架（ドライブ）

文学　ルートディレクトリ

日本文学　外国文学　階層ディレクトリ(1)

古典文学　近世文学　現代文学　古典文学　近世文学　現代文学　階層ディレクトリ(2)

■階層ディレクトリの概念

●絶対指定と相対指定

ディレクトリの指定方法には、

絶対指定

相対指定

の2つがあります。

絶対指定は、ディレクトリ階層のいちばん上のルートディレクトリから順番に下の階層へ指定していく方法です。

先にあげた「B：¥BUNGAKU¥NIHON¥KOTEN」は絶対指定の方法です。

相対指定では、例えば、現在位置しているディレクトリであるカレントディレクトリが、「B：¥BUNGAKU¥NIHON¥KINSE」だったとすると、上と同じ「B：¥BUNGAKU¥NIHON¥KOTEN」のディレクトリを指定する場合、

..¥KOTEN

と指定します。

相対指定は、カレントディレクトリを基準に「自分がいまいるディレクトリから移動したい」ディレクトリを指定する方法です。

ここで「..」は、カレントディレクトリのひとつ上のディレクトリ階層を表わしています。つまりこの場合は、「KINSE」からひとつ上の階層「NIHON」にいったん登り、そこから「NIHON」の下の階層にある「KOTEN」を相対的に指定しています。

¥　ディレクトリの区切りの記号。初めの場合は、ルートディレクトリを示す記号

..　ひとつ上の階層のディレクトリを示す記号

ファイルはディスクにあるプログラムやデータ

フロッピーディスクやハードディスクに記録されているプログラムやデータのことを「**ファイル**」といいます。

ファイルも名まえをつけて管理します。

●ファイル名

ファイル名は、アルファベットやカナ文字（1バイト文字という）など8文

字。それに加えて、「.」(ピリオド)で区切った**拡張子**3文字を組み合わせてつけます。拡張子には、ファイルの性質を表わす3文字が一般に使われています。

慣習的に多くつけられるいくつかの拡張子がありますが、ファイル名をつける人がこれとは別の適切な拡張子をつけることもできます。このファイル名と拡張子の両者を合わせてファイル名という場合もあります。

例えば、次のようなファイル名があります。

SAMPLE. DOC

「SAMPLE」がファイル名です。

「.DOC」が拡張子です。この「DOC」はドキュメント、つまり文書であることを示しています。他にも、

EXE　MS-DOSの実行可能なプログラムファイル
COM　MS-DOSの実行可能なプログラムファイル
BAT　バッチファイル
SYS　システムファイル
DIC　辞書ファイル
BAS　BASICのプログラムファイル
TXT　テキストファイル

などなどがよく使われています。

ファイル名やディレクトリ名には、漢字などの日本語の文字（2バイト文字という）も使うことができますが、名まえには8バイト（漢字などで4文字）＋拡張子3バイトしか使えません。

日本語の文字を使って、長すぎた名まえをつけたりして、誤ったファイルの操作をすると、そのあとファイルを読み出したり、実行できなくなる場合がありますので、日本語の文字を使った名まえは避けたほうがよいでしょう。

●ワイルドカード

ファイル名を指定するときに「ワイルドカード」と呼ばれる記号を使って、複数のファイルをまとめて取り扱うことができます。例えば、

ATOK6A. SYS

ATOK6B. SYS

の2つのファイルを示すのに、

ATOK6 ?. SYS

というように「?」(クエスチョンマーク)を使って示すことができます。

「?」は、「1文字ならば、どんな文字がきてもかまわない」という意味を表わすワイルドカードです。ちょうどトランプのポーカーゲームで、「ジョーカー」の果たす役割に似ています。ジョーカー(ババ)は、「ワイルドカード」とも呼ばれていますので、こういう名まえがついたのでしょう。

ワイルドカードには、「*」(アスタリスク)もあります。これは、「?」よりも便利です。「*」ひとつで、「何文字でも、他の文字がきてもかまわない」という意味の記号です。

例えば、上の2つのファイルを「*」を使って表わすと、

ATOK6 *.*

と表わされてしまいます。いいえ、もっと簡単に

.

としてしまうこともできます。

しかし、こうすると上の2つ以外のファイルがあった場合は、すべてのファイルを指定したことになってしまいます。いかがですか。慣れてくれば便利に活用できるようになります。

●パス

ソフトを起動する際に、いちいち階層ディレクトリのいちばん上から正確にディレクトリを指定していたのではたいへんです。このために「パス」(道筋の意味)と呼ばれるディレクトリを探していく道筋を指定します。

パスを指定しておくと、指定されたディレクトリのソフトが登録されているかどうか、MS-DOSが順番にたどっていき、見つかったところで実行します。

こうしておけば、利用している人は実行するソフトがどのディレクトリに登録されているのか気にしなくてもよくなるわけです。

パスは、「PATH」コマンドを使って指定します。例えば、

PATH A :¥BUNGAKU¥NIHON ; A :¥BUNGAKU¥GAIKOKU ⏎

と、ディレクトリを「;」(セミコロン)で区切って指定すると、2つのディレクトリから実行するファイルを探すようになります。

このコマンドは、一般的にAUTOEXEC. BATのファイルのなかで使われます。

MS-DOSのインストール

実行用ディスクをつくる

オリジナルディスクの事故に備えて

MS-DOSがなければ、アプリケーションソフトを使うことができませんが、購入したMS-DOSのオリジナルをそのまま使っていたのでは、まちがえてフロッピーディスクを壊したり、記録されているファイルを消してしまったりという事故で、たいせつなオリジナルのファイルを失ってしまう場合があります。

そのため、MS-DOSやアプリケーションソフトは、実行用のフロッピーディスクを作成して使用し、オリジナルのディスクはたいせつに保管しておきます。

ここでは、まずソフトの基本となるMS-DOSの実行用フロッピーディスクを作成する「インストール」の作業をします。

まず、MS-DOSのオリジナルのシステムディスクと、まだ使っていない新しいフロッピーディスクを1枚用意します。このとき、誤ってこのたいせつなオリジナルのフロッピーディスクを壊してしまわないように、MS-DOSのフロッピーディスクの切り込み部分にライトプロテクトシールを貼っておきます。

実行用ディスク作成の手順

では、新しいフロッピーディスクをMS-DOSの実行用ディスクにします。

1．PC-9801本体の電源スイッチをONにします。

2．すぐにA：ドライブ（本体のフロッピーディスクドライブで「1」と書かれているドライブ）に「MS-DOSのシステムディスク」を挿入して、ハンドルを時計回りにカチャッというまで回してロックします。

ただし、3.5インチフロッピーディスクドライブの場合は、ハンドルがないので、フロッピーディスクが見えなくなるまできちんと挿入します。

3．ガチャガチャと小さな音がして、少し待つと、MS-DOSが起動したことを示す表示が現れます。

4．つづけて、「日付」や「時間」をたずねる入力要求があります。ここでは、

現在の日付は　19XX-XX-XX(×)です.

日付を入力してください　:⏎

現在の時刻は　XX : XX : XX.00です.

時刻を入力してください　:⏎

のように、それぞれ⏎キーを押します。

5．ディスプレイに

A>

と表示されるので、これにつづいてキーボードから

FORMAT　B :/S⏎

と入力します。

6．FORMATコマンドが起動して、コマンドのバージョン（版）表示の後に、

新しいディスクをドライブB :に挿入し、

どれかのキーを押してください.

と案内が表示されます。

7．案内に従って、用意してある新しいフロッピーディスクをB：ドライブ（「2」と書かれているフロッピーディスクドライブ）に挿入してハンドルを回してロックします。

8．キーボードの⏎キーを押します。

9．フォーマットにとりかかり、ディスプレイに、

ディスクのタイプは 1 :640(KB)　2 : 1(MB)=

と、確認の案内が出ますから、フロッピーディスクの容量に合った数字をキー入力します。

2DD（640KB）の場合……1

2HD（1 MB）の場合……2

10．フォーマットの作業に入り、2HDの場合はディスプレイに、

目的のディスクは1MB FDです

フォーマット中です. . .

と、表示されて、フロッピーディスクにカチャカチャとアクセスしてフォーマットが行われます。

11. フォーマット作業は、少し時間がかかりますが、終了すると、

目的のディスクは1MB FDです

フォーマット中です．．．フォーマットが終了しました

と、表示が変わり、つづいて、MS-DOSのシステムの転送作業を行います。

システムを転送しました

と表示されて、フォーマットとシステムのすべての作業が終了します。

12. 作業が終了すると、

別のディスクをフォーマットしますか<Y/N>？

と、次の別のディスクのフォーマットをつづけるかどうかをたずねてきます。ここでは、他のディスクのフォーマットを行わないので、キーボードから、N⏎と、入力してすべての作業を終了します。

するとまた、MS-DOSのコマンドの入力待ちの状態であるA>に戻ります。

これで、B：ドライブのフロッピーディスクがMS-DOSのシステムフロッピーディスクになりました。

ディスクの内容を見る

できあがったシステムフロッピーディスクの内容を見てみましょう。

プロンプト「A>」につづいて、キーボードから、**DIR　B：**⏎と、入力すると、ディスプレイに

ドライブB：のディスクのボリュームラベルは、ありません

ディレクトリはB：¥

COMMAND　COM　23942　85-12-11　15：28

1個のファイルがあります．

1164288バイトが使用可能です．

のように表示されて、システムとともに「COMMAND.COM」のファイルが転送されていることがわかります。ここに表示されている数字は、それぞれフォーマットに使用したシステムによって違っていますが、これでMS-DOSのシステムフロッピーディスクができあがったことが確認できます。

ここに示した方法は、41ページのフロッピーディスクのフォーマットを行う手順の詳細ですが、Quick BASICの実行用フロッピーディスクを作成するばかりではなく、別のアプリケーションソフトの実行用フロッピーディスクを作成する方法でもありますので、参考にしてください。

MS-DOSのコマンド

MS-DOSには、ファイルを操作するために必要なコマンドが用意されています。これらのコマンドのうち、いくつかの基本的なものの使い方を簡単に説明しましょう。

DIR

ディレクトリのなかに登録されているファイルの一覧を表示します。

DIR A：⏎

とキーボードから入力すると、例えば、

```
A>DIR A:

 ドライブ A: のディスクにはボリュームラベルがありません
 ディレクトリは A:¥

COMMAND  COM    17190  83-09-19  16:19
AUTOEXEC BAT       73  90-01-01   8:59
CONFIG   SYS      126  90-01-01   8:59
ATOK6A   SYS    54353  88-08-11  12:00
ATOK6B   SYS    21228  88-08-11  12:00
BIN         <DIR>      90-01-01  17:17
LIB         <DIR>      90-01-01  17:17
        7 個のファイルがあります
    98304 バイトが使用可能です
```

ファイル名　拡張子　バイト数　日付　時間

上のようにAドライブに登録されているファイルやディレクトリが表示されます。

バイト数の項目が「<DIR>」となっている行は、ファイル名ではなく、ディレクトリであることを示しています。

ドライブ名につづいてディレクトリ名を指定すれば、指定したディレクトリ内のファイルの一覧を表示させることができます。

REN (RENAME)

ファイルの名まえを変更するコマンドです。
「README. DOC」を「YONDENE. BUN」に変更してみましょう。

REN　README. DOC　YONDENE. BUN⏎

これで、ファイル名が変更されました。
DIR コマンドで確かめてみましょう。

DEL (ERASE)

指定したファイルをフロッピーディスクやハードディスクから削除するコマンドです。
一度削除してしまうと、もとに戻すことができませんので注意しましょう。

DEL　SAMPLE. DOC⏎

とすると、カレントディレクトリの「SAMPLE. DOC」というファイルが削除されます。

TYPE

テキスト形式のファイルの内容を表示します。
Quick BASIC の場合、プログラムは「標準ファイル」と「テキストファイル」の 2 つの形式で保存する方法があります。
このうち、テキストファイルは文字として登録されているので、このコマンドでディスプレイに表示させて読むことができます。
標準ファイルや EXE 指定を行ったプログラムファイルなど、文字として登録されていないファイルは、表示が乱れたりして読むことができません。

TYPE　A：README.DOC⏎

とすると、A：ドライブの「README.DOC」というテキスト形式のファイルの内容を表示することができます。

COPY

ファイルをコピーするコマンドです。

COPY　A：README. DOC　B：⏎

とすると、A ドライブの「README. DOC」が、B ドライブにコピーされま

す。ファイル単位ばかりではなく、ワイルドカードを使って、

COPY　A：¥BUNGAKU¥　＊．＊　A：¥BUNGAKU¥GAIKOKU⏎

とすると、Aドライブの「BUNGAKU」のディレクトリ内のファイルがすべて「BUNGAKU¥GAIKOKU」にコピーされます。

FORMAT

MS-DOSのフォーマットで、フロッピーディスクやハードディスクを初期化します。

新しいフロッピーディスクなどは、ちょうどなにも書かれていない紙のようなものです。これをページごとに切ったり、けい線をひいたり、束ねてノートにするような作業をして、初めてデータを記録したり、読み出すことができるようになります。このような作業がフォーマット（初期化）です。

新しいフロッピーディスクを使うときには、必ず初めにフォーマットをしなければなりません。

データディスクをつくるには、AドライブにMS-DOSのシステムフロッピーディスクを入れ、Bドライブに新しいフロッピーディスクを入れて、

FORMAT　B：⏎

とします。

フォーマット作業が始まると、フロッピーディスクのタイプを質問してきます。2DDなら「1」、2HDなら「2」をキーボードから入力して答えます。

フォーマットコマンドを実行するときは、Bドライブに新しいフロッピーディスクを入れて、コマンドの後ろに必ずBドライブを指定するようにしましょう。そうしないと、誤って他のドライブのフロッピーディスクをフォーマットして、貴重なプログラムやデータを失ってしまう場合があります。

MS-DOSのシステムディスクをつくるときには「／S」スイッチを指定します。

FORMAT　B：　/S⏎

とすると、Bドライブのディスクをフォーマットし、自動的にシステムを転送して、MS-DOSのシステムディスクが作成されます。

MD (MKDIR)

新しくディレクトリをつくります。

ルートディレクトリの下にディレクトリをつくったり、下の階層のディレクトリのさらに下にディレクトリをつくります。ディレクトリの下のディレクトリを「サブディレクトリ」という場合があります。

MD　B：¥BUNGAKU¥NIHON⏎

とすると、Bドライブにあるフロッピーディスクの「BUNGAKU」のディレクトリの下に、「NIHON」というディレクトリをつくります。

CD (CHDIR)

カレントディレクトリを他のディレクトリに移します。カレントディレクトリから他のディレクトリに移動するといったほうが感覚に合うかもしれません。

カレントディレクトリが、B：¥BUNGAKU¥NIHONの場合、

CD　B：¥BUNGAKU¥GAIKOKU⏎

で、「NIHON」から「GAIKOKU」にディレクトリを移動したことになります。また、ディレクトリの相対指定を使うと、

CD　..¥GAIKOKU⏎

の指定で、一気に「GAIKOKU」のディレクトリに飛んでしまいます。

RD (RMDIR)

不要になったディレクトリを削除します。ただし、削除しようとしたディレクトリのなかにファイルがあったり、さらに下層のディレクトリがあった場合は、削除することはできません。不要なディレクトリに属するすべてのファイルやディレクトリを削除したあとに削除します。

RD　B：¥BUNGAKU¥NIHON¥KINSE⏎

とすると、「KINSE」のディレクトリが削除されます。

いったん削除されたディレクトリは、もとに戻すことはできません。もう一度MDコマンドでつくり直します。

RENDIR

一度作成したディレクトリの名まえを変更します。

RENDIR　B：¥SAMPLE　¥MIHON⏎

とすると、Bドライブの「SAMPLE」というディレクトリ名が、「MIHON」というディレクトリ名に変更されます。

ただし、MS-DOSのバージョンによっては、このコマンドがないものもあります。

★コマンドの検索★

MS-DOSでは、キーボードから文字列が入力されると、COMMAND.COMが働いて、文字列がコマンドかどうかを調べます。

このときに、初めに入力された文字列が内部コマンドならば、すぐに指定されたコマンドを実行します。それ以外のときには、フロッピーディスクやハードディスクに登録されている外部コマンドのなかから、指定された文字列と同じファイル名のコマンドがないかを検索します。

このとき、拡張子によって、

「.COM」⟶「.EXE」⟶「.BAT」

の順に検索されて、初めにファイル名が一致したファイルを、コマンドとして実行します。このために、Quick BASICなどで作成した独立実行型プログラム（.EXEファイル　P.130参照）をMS-DOSのコマンドと同じように直接実行できるわけです。

ただし、拡張子の優先順位の高い同じファイル名の外部コマンドがあった場合は、目的のプログラムとは別のコマンドが実行されてしまいますので、注意してファイル名をつけなければなりません。

また、外部コマンドの検索は、カレントディレクトリのなかでのみ行われるので、実行したいコマンドが他のディレクトリにある場合は、ディレクトリとコマンド名をいっしょに指定しなければなりません。

どのディレクトリからでも、カレントディレクトリにかかわらず外部コマンドやプログラムを実行したい場合は、PATHコマンド（P.19参照）を使って、外部コマンドやプログラムの登録されているディレクトリにパスを設定しておきます。

バッチファイル

決まった手順のコマンドはひとまとめにして

パソコンを使っていると、ある一連のコマンドを順番に入力していく場合がよくあります。

例えば、カレントドライブとカレントディレクトリを移動して、ユーティリティプログラムを動かしてから、アプリケーションソフトを起動するなどは、だれでも日常的に行っていることでしょう。

このように決まりきった手順を、毎回キーボードから入力するのはたいへんですし、まちがいも起こりやすくなります。

MS-DOSには、こうした一連のコマンドを、実行順に登録してファイルにしておくと、このファイル名をキーボードから打ち込むだけで、指定された順番にコマンドを実行する機能が備えられています。

この実行手順を登録したファイルを「**バッチファイル**」といい、ファイル名を指定して実行することを、「**バッチ処理を行う**」などといいます。

バッチファイルは、MS-DOSのコマンドを実行順に並べただけのテキストファイルですので、エディタやワープロソフトで簡単に作成することができますが、ファイル名の拡張子には必ず「.BAT」をつけなくてはなりません。

バッチファイルのなかでも、AUTOEXEC. BATという特別なものがあります。オートエクシキュート．バッチ（自動実行）ファイルです。

Quick BASICのインストールでもつくられますが、MS-DOSが起動したときにいちばん最初に実行されるファイルです。

パソコンやソフトを使用する環境の設定などを行い、つづいてソフトをオートスタートしたりするように指定したファイルがつくられています。

このファイルも簡単につくることができますが、自動実行ファイルとするためにはファイル名を必ず「AUTOEXEC. BAT」としなければなりません。

PART 1

レッツ Quick BASIC

Quick BASIC の特徴は

いままでのBASICとくらべると

わかりやすいプログラムをマウスを使って

PC-9801シリーズには、本体の部品の一部であるROM※に記録されたBASIC「N₈₈-BASIC(86)」と、フロッピーディスクに記録された「N₈₈-日本語BASIC(86)」の2つのBASICが標準で付属してきます。

N₈₈-BASIC (86) は、ディスクドライブにフロッピーディスクを入れずに本体のスイッチをオンにすると自動的に起動します。

N₈₈-日本語BASIC(86)は、付属のシステムディスクをディスクドライブに入れて、スイッチをオンにして起動します。

N₈₈-日本語BASIC(86)は、N₈₈-BASIC (86) にくらべて、

フロッピーディスクの制御
日本語文字列の入力と操作
拡張グラフィック

などの機能が拡張されています。

フロッピーディスクやハードディスクなどが普及してきた現在では、「標準」のBASICというと、一般的にはN₈₈-日本語BASIC (86) のことをさします。

この標準BASICは、プログラムのファイル操作などをするコマンドや、プログラムを作成するエディタ、テストランをしてプログラムのまちがいを修正するデバッグの機能を備えたBASICです。

すべて、キーボードから入力するコマンドで操作しますが、BASICの仕様が

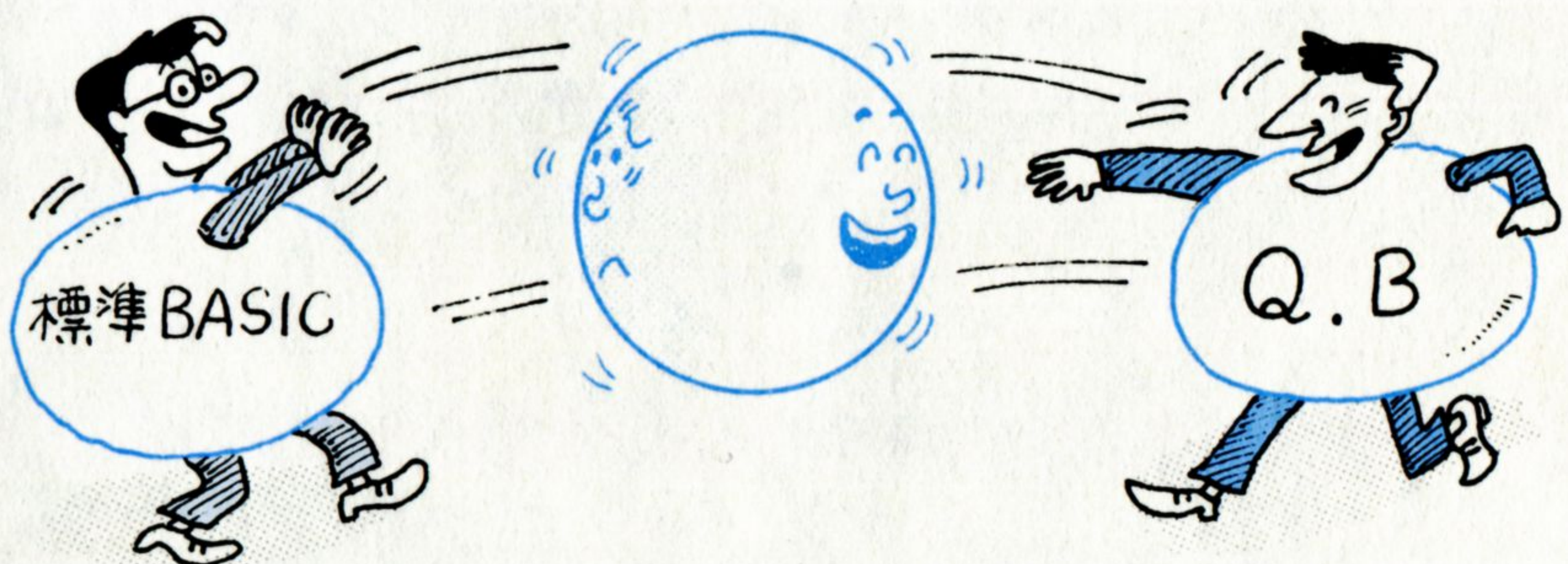
ROM
標準 BASIC
N88-BASIC(86)
N88-日本語 BASIC(86)
コンパイル
インタプリタ

これらの機能と混然一体となっているために、操作に慣れにくくしています。

また、BASIC 自体が構造化されていないので、つい未整備でわかりにくいプログラムができあがったりします。

これにくらべて、Quick BASIC は、エディタやデバッグ、プログラムの実行といった機能がそれぞれ整然と整理され、マウスを使って操作しやすくなっています。また、BASIC も構造化されて、わかりやすいプログラムが書きやすくなっています。

しかも、コンパイル※というパソコンが直接実行できるマシン語のプログラムに変換する機能があるので、実行速度も、いままでのインタプリタ※より、数倍から百倍近くも早く実行することができるようになりました。

次にその特徴をいくつかあげてみましょう。

※ROM　パソコンに組み込まれている部品で、データを記録する半導体素子。記憶装置の一部でメモリと呼ばれています。ROM(ロム)は、Read Only Memory の略で、データを読み出すことしかできないメモリのことです。

これに対して、データを書き込んだり（記録・記憶）、読み出したりすることができるメモリをRandom Access Memory、略してRAM（ラム）と呼んでいます。

※コンパイル　プログラムの実行前に、パソコンが直接実行できるマシン語のプログラムに一括して変換すること。

プログラムの実行速度が早くなりますが、変換前のソースプログラムでないと、プログラムの修正ができないため、プログラムのデバッグも難しくなります。一般に変換には手間がかかり、十分なMS-DOS の知識と経験が必要です。

※インタプリタ　プログラムの命令をひとつずつ解釈・実行していく方法。

プログラムの実行速度は遅くなりますが、ソースプログラムのまま実行や修正ができるので、試行錯誤によってプログラムを作成していくことができるなど、初心者に向いています。

親切な画面

●マウス

Quick BASICでは、入力装置にマウス※を使うことができます。カーソルの移動やコマンドの選択は、すべてマウスで行うことができるので、マウスを使えば、コマンドなどをキーボードから1文字ずつ打ち込む必要はなくなります。

また、編集する範囲や画面のスクロールも、マウスを使って指定することができます。

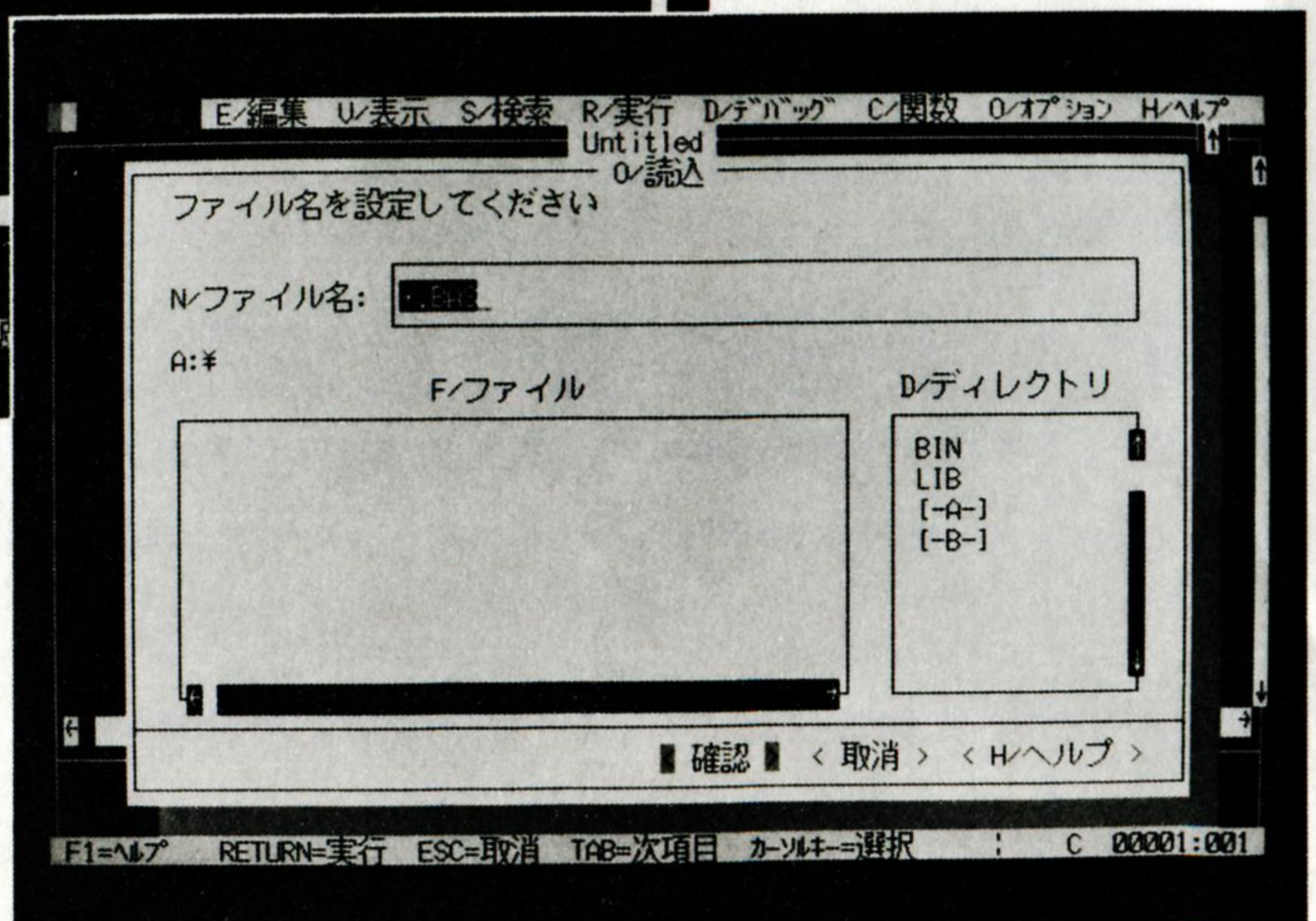
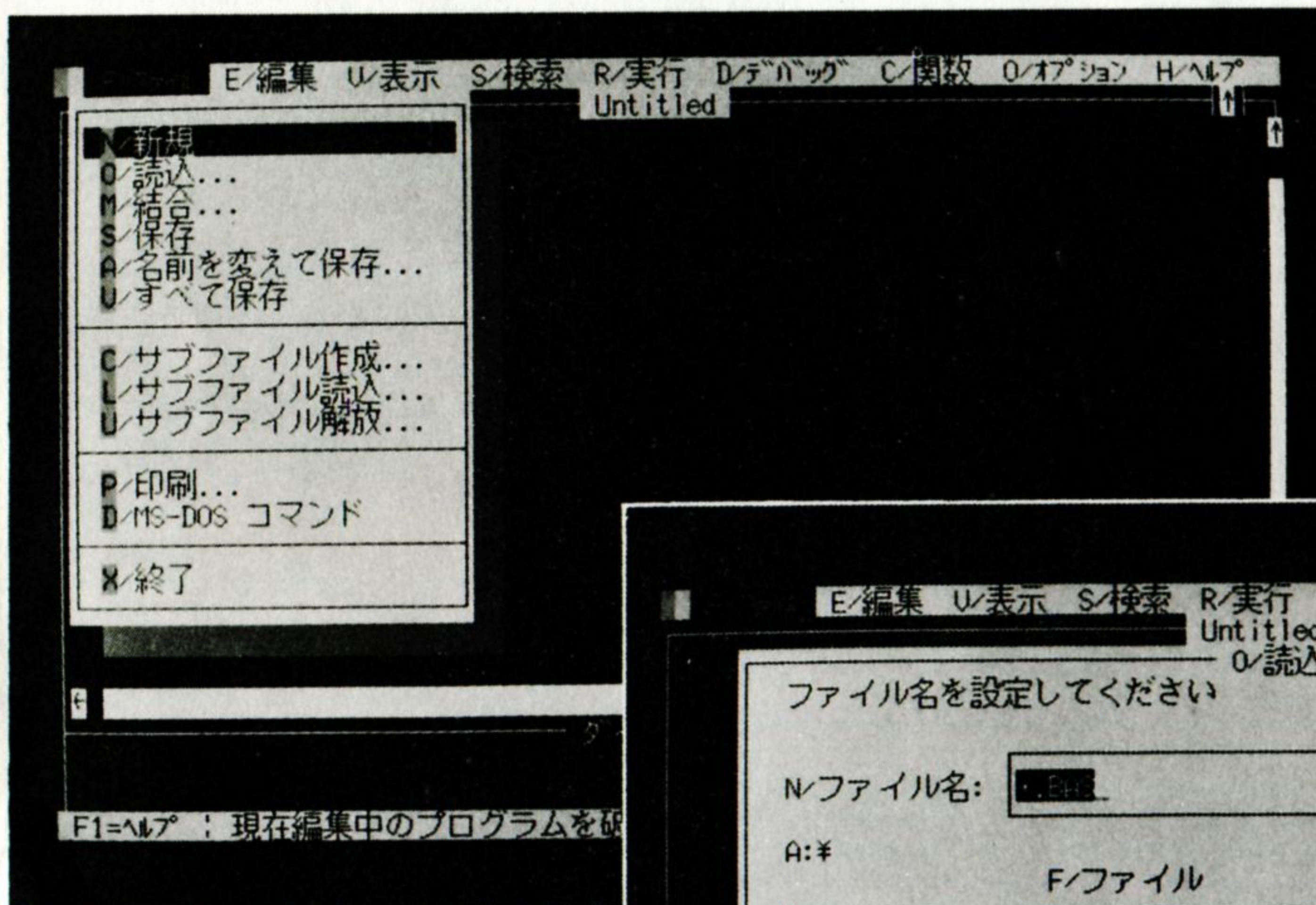
もちろん、マウスがなくても操作できますが、操作性はかなり落ちますので、マウスを使ったほうがよいでしょう。

●プルダウンメニュー

コマンドの選択には、プルダウンメニューを使うことができます。編集やファイルの操作を選択すると、使うことのできるコマンドの一覧表が、画面の上から下りてくるように表示されてきます。この一覧のなかから必要な作業をす

■プルダウンメニューの画面

■ダイアログボックスの画面

マウス
クリック
QBアドバイザ　ダイアログボックス

るコマンドを選んで、マウスでクリック※すれば、すぐに作業ができます。

また、現在選択されているコマンドや、使うことのできないコマンドがわかるようになっています。

細かい指定が必要な場合は、ダイアログボックスと呼ばれる対話のための窓が開き、詳細な設定を求めてきますし、設定の確認や取り消しも行うことができます。

●QBアドバイザ

キーボードから[f・1]または[SHIFT]＋[f・1]キーを押せば、いつでもヘルプ画面を見ることができます。QBアドバイザは、コマンドやBASICのキーワードの説明、BASICの文法などを面画に表示する機能です。QBアドバイザを使えば、マニュアルを引く手間が省けます。

●エディタ機能

プログラムを書くために各種の便利な機能が用意されています。

プログラムの一部をコピーしたり、ほかの場所に移動したりできますから、同じことを何度もタイプしなくてもすみます。まちがえて削除してしまった場合にも、取り消すことのできるアンドゥ機能も備えています。特定の文字列を探し出す検索や、文字列の置き換えをする置換の機能を使うことができます。

また、プログラムのケアレスミスをチェックするための、構文チェックもできます。

●デバッグ機能

プログラムのまちがいを探し出す手助けをするのが、デバッグ機能です。

プログラムで使われている変数の値をチェックしていくことのできるウォッチや、プログラムの実行をステップごとに追っていくトレース、これを逆順で行うヒストリの機能があります。プログラムが一定のところにくると実行を止

※マウス　形がネズミに似ているのでこう呼ばれる入力装置。机の上に置いて手で動かすと、その動きに従ってパソコン画面上のマウスカーソルが動き、ボタンを押してコマンドの選択をするなどの作業を進めます。

※クリック　マウスのボタンを１回だけ押すことをクリックするといいます。２回押すことをダブルクリック、押したままマウスを移動することをドラッグするといいます。

める、ブレークポイントも設定できます。

また、サブプロシージャを実行しているときに、プロシージャ[※]の階層を表示したり、指定した階層までの実行をすることもできます。

●構造管理

プログラムの論理構造を一覧表示したり、そのなかからプロシージャを呼び出したりすることができます。

新しいプロシージャを定義する場合には、定義を始めるとすぐに新規のプロシージャとして管理を始めます。

使いやすい統合環境

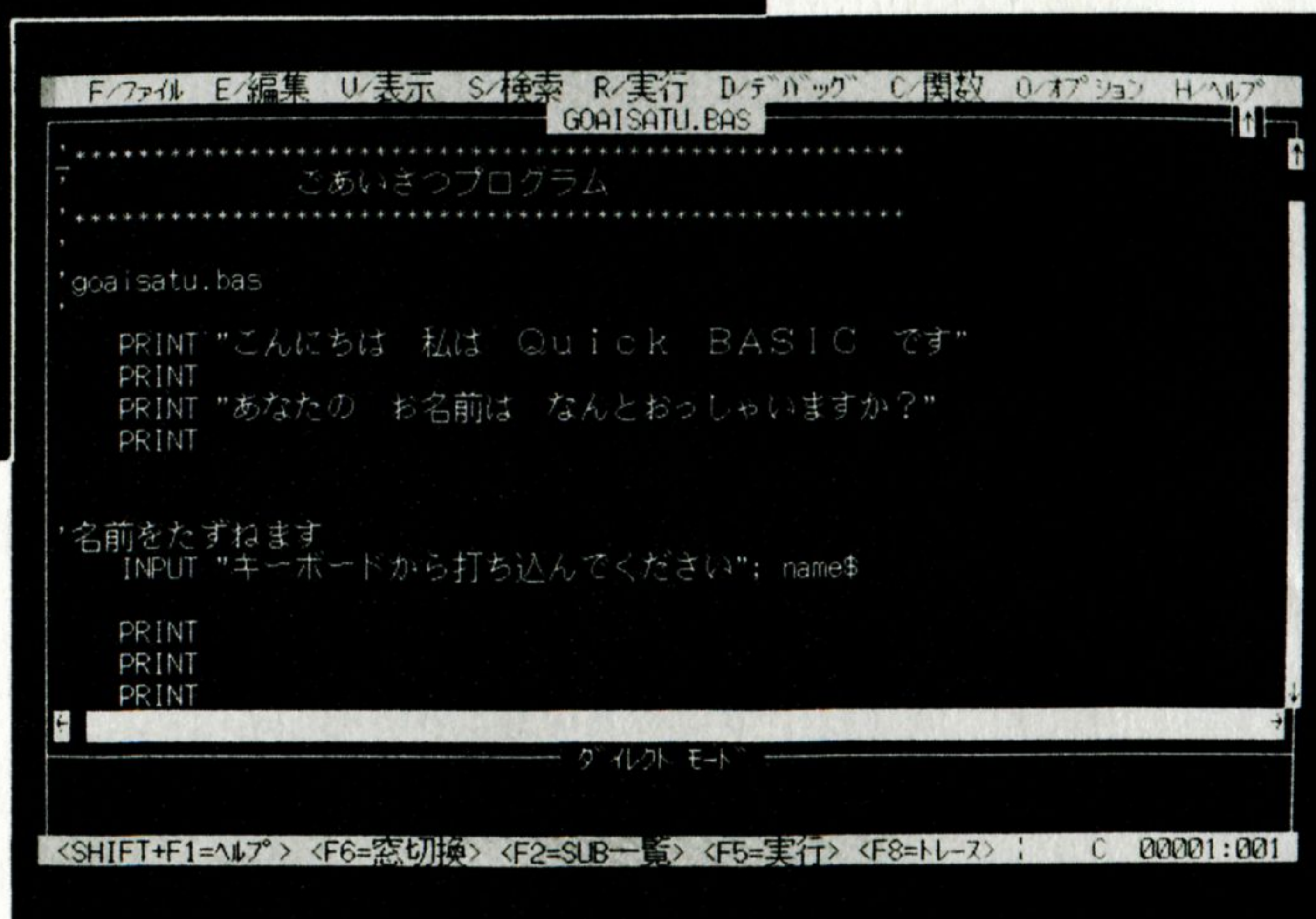
●エディタ画面と実行画面

Quick BASICでは、プログラムを作成・編集するエディタの画面と、プログラムの実行画面が明確に分かれています。

エディタ画面には、プルダウンメニュー用のメニューバーや、現在編集対象

```
こんにちは　私は　Ｑｕｉｃｋ　ＢＡＳＩＣ　です
あなたの　お名前は　なんとおっしゃいますか？
キーボードから打ち込んでください? HIROMI

HIROMI　さん　こんにちは！
これからも　よろしくお願いしまーす！！

何かキーを押してください
```

■実行画面

```
F/ファイル  E/編集  V/表示  S/検索  R/実行  D/デバッグ  C/関数  O/オプション  H/ヘルプ
                         GOAISATU.BAS
'*************************************************
'                 ごあいさつプログラム
'*************************************************
'
'goaisatu.bas
'
    PRINT "こんにちは　私は　Ｑｕｉｃｋ　ＢＡＳＩＣ　です"
    PRINT
    PRINT "あなたの　お名前は　なんとおっしゃいますか？"
    PRINT

'名前をたずねます
    INPUT "キーボードから打ち込んでください"; name$

    PRINT
    PRINT
    PRINT
------------------------ ダイレクト モード ------------------------

<SHIFT+F1=ヘルプ> <F6=窓切換> <F2=SUB一覧> <F5=実行> <F8=トレース> ¦   C  00001:001
```

■エディタ画面

になっているプログラムの名まえが表示され、機能的にプログラムの作成ができるようになっています。

また、マウスで効率的に画面の操作ができるようにゲージの入ったスクロールアイコンやスクロールボックスも表示されています。

プログラムが実行されると、画面が切り替わって、実行画面になります。

こちらは、実際にプログラムが実行されている画面で、エディタ画面から実行画面に戻ると、それまでに実行して表示されていた画面がそのまま残っています。

表示切り替えの機能を使えば、エディタ画面と実行画面を自由に往復して、混乱なく見ることができます。

●ダイレクトモード

エディタ画面の下にダイレクトモードのウィンドウがあります。

ダイレクトモードは、Quick BASICのステートメントを試しに実行してみるとか、プログラム作成中に画面での表示位置を確認するとかのテストを行うことができます。

このときの結果は実行画面に表示されますので、プログラム中のエディタ画面の乱れを気にする必要はありません。

また、プログラム中で使われている変数に任意の値をセットして、プログラムのデバッグに役立てることができます。

●モジュールとファイルの管理

Quick BASICのプログラムは、複数のモジュールからなりたっています。

モジュールはメインモジュールを先頭に階層的に管理され、メニューバーから簡単に一覧表示させたり、新しいモジュールを定義することができます。

また、実行したり編集対象になっているモジュールは常に把握されていて、タイトルバーにモジュール名が明示されています。

ファイルは、複数のモジュールをひとつのプログラムとして一体として管理

※プロシージャ　いくつかの命令でできているひとつのサブルーチンで、通常はCALLステートメントや式のなかで呼ばれ、END SUB、END FUNCTIONで終了します。

され、フロッピーディスクに保存したり、読み出したりすることができます。

現在編集中のプログラムの途中に、ほかのプログラムを読み出したりもできますので、プログラムの再利用もできますし、大きなプログラムを分割して作成していくこともできます。

ファイル形式は、Quick BASICの標準形式だけでなく、MS-DOSのテキスト形式でも保存、読み出しができますので、ほかのエディタやユーティリティを使って効率よくプログラムの開発や管理ができます。

構造化されたBASIC

行番号のいらないわかりやすいプログラム

●行番号

Quick BASICには行番号がありません。プログラムは書かれている順番に、上から下へと実行されます。

プログラムの分岐やループは、構造化された構文を使って行われます。また、プログラムのジャンプは、ラベルを使って行われ、行番号のないわかりやすいプログラムを書くことができます。

●構造化プログラム

プログラムをつくるときには、ひとつの作業をする単位ごとにつくっていくのが、最もつくりやすく、かつ理解しやすいものです。

■行番号のないQuick BASICのプログラム例

```
DO
    Chuusi = Iie
    Bangou = 0
    DO
        Bangou = Bangou + 1
        RecordNumber = Bangou
        DATA.YOMIKOMI DATA$(), RecordNumber
        PRINT "------"; Bangou; "------"
        FOR i = 1 TO KoumokuSuu
            PRINT Koumoku$(i); " : ";
            PRINT DATA$(i)
        NEXT
        PRINT
        Kari$ = INPUT$(1)
        IF LCASE$(Kari$) = "e" OR Kari$ = "ｲ" THEN Chuusi = Hai
        IF SaidaiBangou <= Bangou THEN Chuusi = Hai
    LOOP UNTIL Chuusi = Hai
LOOP UNTIL LCASE$(Kari$) = "e" OR Kari$ = "ｲ"
```

Quick BASICでは、この単位をモジュール、サブプログラム、ユーザー定義変数などのブロックごとに作成し、管理するという方法で構造化しています。

また、ひとつのブロックのなかで、ブロック化IF文やSELECT〜CASE文、DO〜LOOP文などの構文を使って、ブロック内のプログラムを構造化することができます。

いままでのBASICが、GOTO文や1行しか使えないIF文のためにわかりにくいプログラムになっていたのにくらべ、きれいで読みやすいプログラミングができるようになります。

プログラム
モジュール
プロシージャ
プロシージャ
モジュール
プロシージャ
プロシージャ

■構造化プログラム

●変数

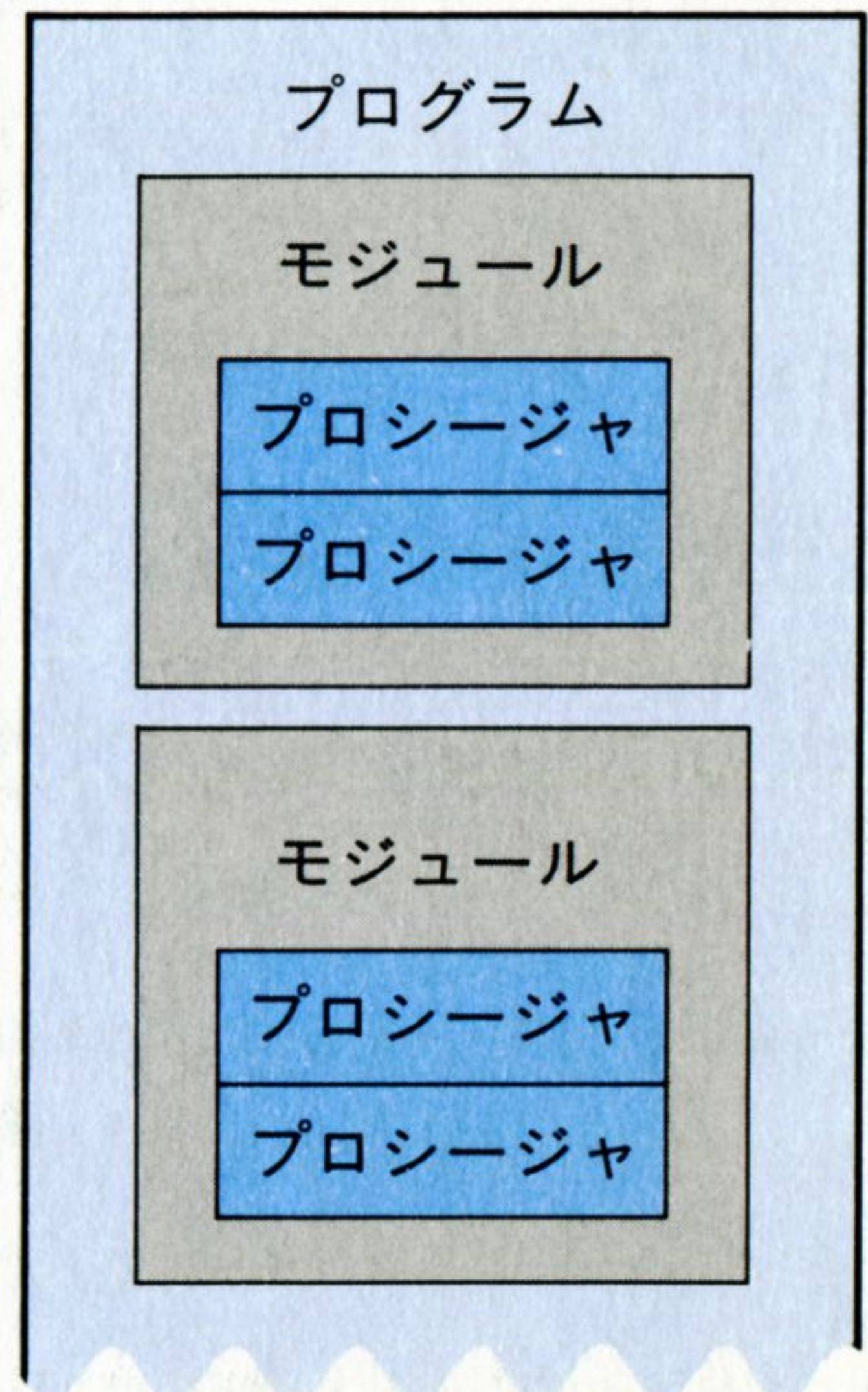
Quick BASICでは、変数の基本は、プロシージャ単位で管理されるローカル変数です。

プログラム全体で共用するには、グローバル変数として特に宣言しなければなりません。

これは、プロシージャが違えば同じ変数名を使用しても、まったく別の変数として取り扱うことができるということです。たくさんのサブプログラムをつくったときでも、変数表などをつくって自分で変数の管理をする必要がなくなります。

●長整数

整数型のデータの場合、標準BASICでは±3万程度までの値しか扱うことができませんでした。これ以上のデータを扱う場合は、浮動小数点のデータで6〜7桁の有効精度を使っていましたが、これでは誤差の修正をしなくてはならないので、ループカウンタなどの計算に使うには不便でした。

Quick BASICでは、長整数を使って±20億程度の計算ができ、この不便さを解消しました。それほど規模の大きくない計算では、長整数を使って、誤差のない計算ができるようになっています。

●ユーザー定義変数

Quick BASICでは、変数の型を任意に定義できるユーザー定義変数が用意されています。

文字や数値などのデータ型を組み合わせて、新たなデータ型の変数としてユーザーが定義すれば、ひとまとまりのデータとして取り扱うことができます。特に、ランダムファイルを使う場合に、レコード単位でデータを変数として取り扱うことができるので便利です。

高度な活用の可能性

マウスのクリックひとつでプログラムのコンパイル

●従来のBASICとの互換性

Quick BASICは、従来のBASICで作成されたプログラムも利用することができます。標準のBASICの場合は、MS-DOSのユーティリティなどを使って、MS-DOSのテキストファイル形式に変換しておく必要があります。

標準BASICとQuick BASICとでは、ステートメントやラベルの使い方が違うなどの互換性のない部分があります。また、行番号はQuick BASICでは、各行につけられたラベルとみなされますので、あってもなくてもかまいません。

単純なプログラムなら、標準BASICのプログラムをそのままQuick BASICで実行することも可能ですが、ほとんどのプログラムは、Quick BASICの仕様に合わせて変更しなければならないでしょう。

いままで使っていたプログラムすべてをつくり直すことを考えれば、この互換性はありがたい機能といえるでしょう。

●コンパイル

Quick BASICは、インタプリタの感覚で、試しにプログラムを動かしながらプログラムの作成ができますので、プログラム開発にはたいへんに便利です。

しかし、プログラムをひとつひとつの命令ごとに解釈して、パソコンにわかるマシン語に変換して実行しているので、この解釈と変換のためのプログラム（これもBASICが行っています）がないと実行することができません。

また、プログラムを実行するときに解釈と変換の過程が入りますので、実行速度も遅くなってしまいます。

これにくらべてコンパイルされたプログラムは、そのままパソコンが実行で

きるマシン語の形式になっています。高速になりますが、実行する前にコンパイルというプログラムを一括して変換する作業が必要になります。

一般のコンパイル作業はかなり複雑です。ある程度パソコンやMS-DOSの知識がないと、難しいでしょう。

コンパイル後にプログラムの誤りを探したり、誤りを修正するためには、変換前のプログラム（ソースプログラム）を修正してから、再度コンパイルしなければならないので、インタプリタにくらべてプログラムの作成に手間がかかります。

Quick BASICは、このコンパイルの作業をマウスのクリックひとつで自動的に行ってくれますので、非常に手軽にコンパイルされた高速なプログラムを作成することができます。

さらに、プログラムを作成中には、Quick BASICの環境下でプログラムをインタプリタで動かすことができますので、プログラムの開発や修正も簡単に行うことができます。

インタプリタと、コンパイラのいいところをひとつに合わせて、統合化された環境にしたのがQuick BASICです。

●クイックライブラリと他の言語との接続

Quick BASICは、自分で作成したプログラムをBASICのサブプログラムや関数として組み込むことができます。

自作のプログラムでQuick BASICの機能を強化することもできますし、プログラムの仕様を共通化させることによってグループで大きなプログラムを開発することもできます。

クイックライブラリは、Quick BASICの高度な利用になりますが、作成の手続きはメニューから選んで簡単に保存、読み出しをすることができます。

また、他言語のマイクロソフトCやマイクロソフトMASMで作成されたルーチンを、Quick BASICのライブラリとして活用することができるようにもなっているので、Quick BASICは拡張性の高い言語であるといえます。

スタートの準備はOK？

初めに1回だけする「インストール」

Quick BASICを実際に使用するためには、初めに1回だけ「インストール」という作業をしなければなりません。

バックアップ　Quick BASICのフロッピーディスクをそのまま使うと、誤操作や取り扱いのミスで、フロッピーディスクを壊してしまったりします。MS-DOSのCOPYコマンドで「バックアップ」というフロッピーディスクのコピーをとっておきます。そして、ふだんはこのフロッピーディスクを使い、マスターディスクはたいせつに保管しておきます。

MS-DOSの用意　Quick BASICには、このプログラムを動かすための基本ソフトであるMS-DOSのシステムが付属していません。MS-DOSを持っていない人は、MS-DOSを別に購入する必要があります。

しかし、「一太郎」などのアプリケーションソフトのなかにはMS-DOSのシステムが付属しているものがあります。このアプリケーションソフトを動かしているシステムだけを使って、Quick BASICを動かすのに利用することもできます。

ワークディスクづくり　Quick BASICを使うためには、インストールを行ってワークディスクをつくる必要があります。

まず、MS-DOSのシステムディスクと新しいフロッピーディスクを、2HDの場合は3枚、2DDの場合は4枚用意します。

MS-DOSの準備

MS-DOSのシステムの入っているフロッピーディスクをAドライブに入れて、PC-9801のスイッチをONにし、すぐにフロッピーディスクドライブのレバーを下ろします。するとMS-DOSが起動して、メニュー画面が表示されます。

ここでは CTRL +Cを押して、メニュー画面からのコマンドの選択を中止します。ディスプレイの画面には、プロンプト「A＞」が表示されてコマンド入力待ちの状態になります。

アプリケーションソフトのMS-DOSを利用する場合は、ソフトを起動して、すぐに終了します。MS-DOSの場合と同じように、ディスプレイの画面には「A＞」のプロンプトが表示されて、コマンド入力待ちの状態になります。

フロッピーディスクのフォーマット

Bドライブに新しいフロッピーディスクを入れて、フロッピーディスクドライブのレバーを下ろしておきます。そして、プロンプト「A＞」の位置から、

FORMAT　B：　/S ⏎

と入力します。

MS-DOSのフォーマットコマンドが起動して、Bドライブのフロッピーディスクをシステムディスクとして使えるように、フォーマット作業が始まります。

MS-DOSのバージョンによって、注意のメッセージなどの細かい部分が違いますが、メッセージに従ってキーを押して作業を進めます。

最後の「別のディスクをフォーマットしますか〈Y／N〉」というメッセージには「N」と ⏎ キーを押して、フォーマットの作業を中止します。

これで、BドライブにMS-DOSのシステムの入った起動ディスク用のフロッピーディスクができました。

次は、アドバイザディスク、ワークディスク用のフロッピーディスクをフォーマットします。

Bドライブのフロッピーディスクを新しいものに入れ替えて、プロンプト「A＞」の位置から、

FORMAT　B：↵

と入力します。

2HDのフロッピーディスクは2枚、2DDの場合は、3枚のフロッピーディスクを同じようにフォーマットしておきます。

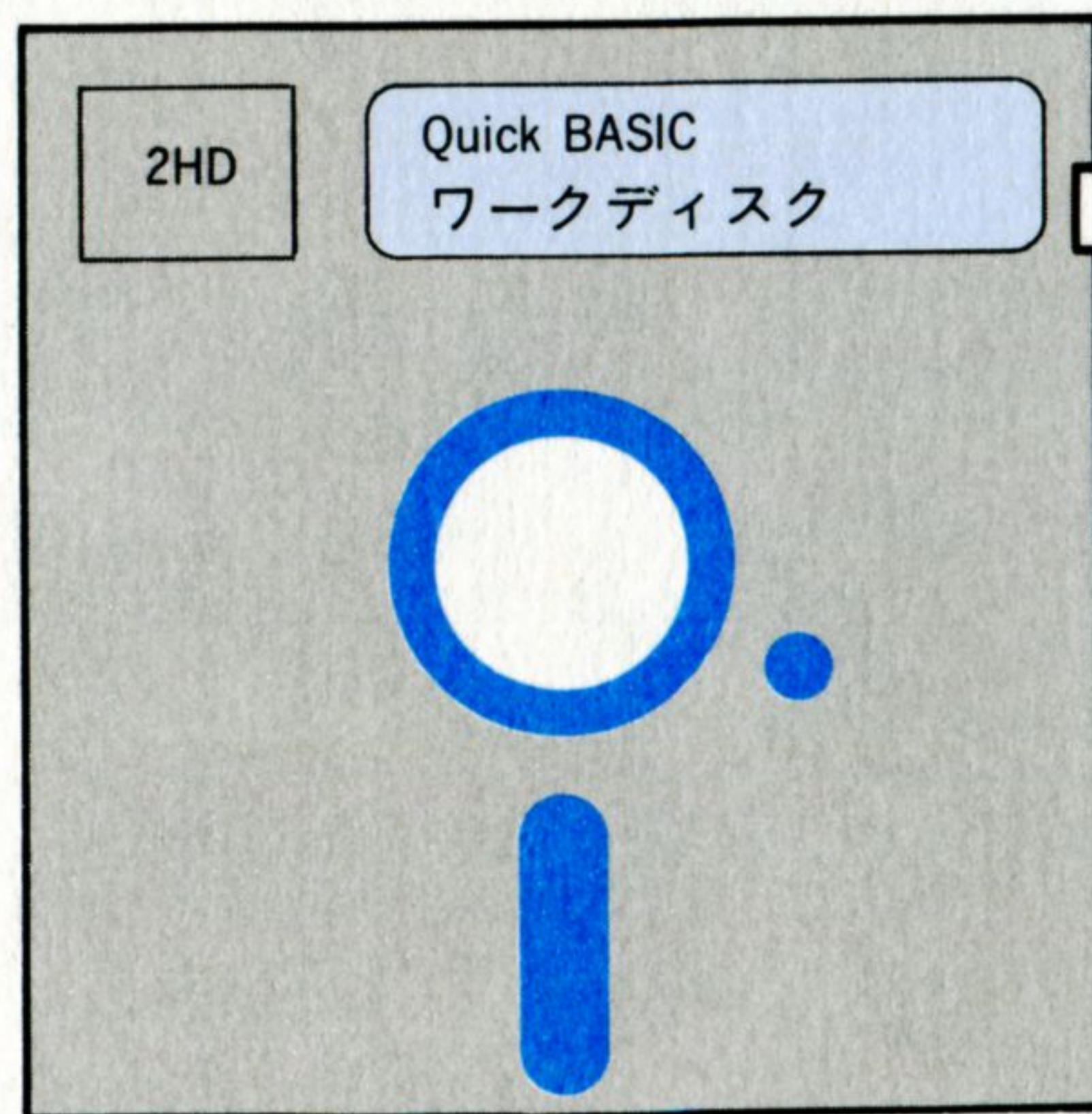
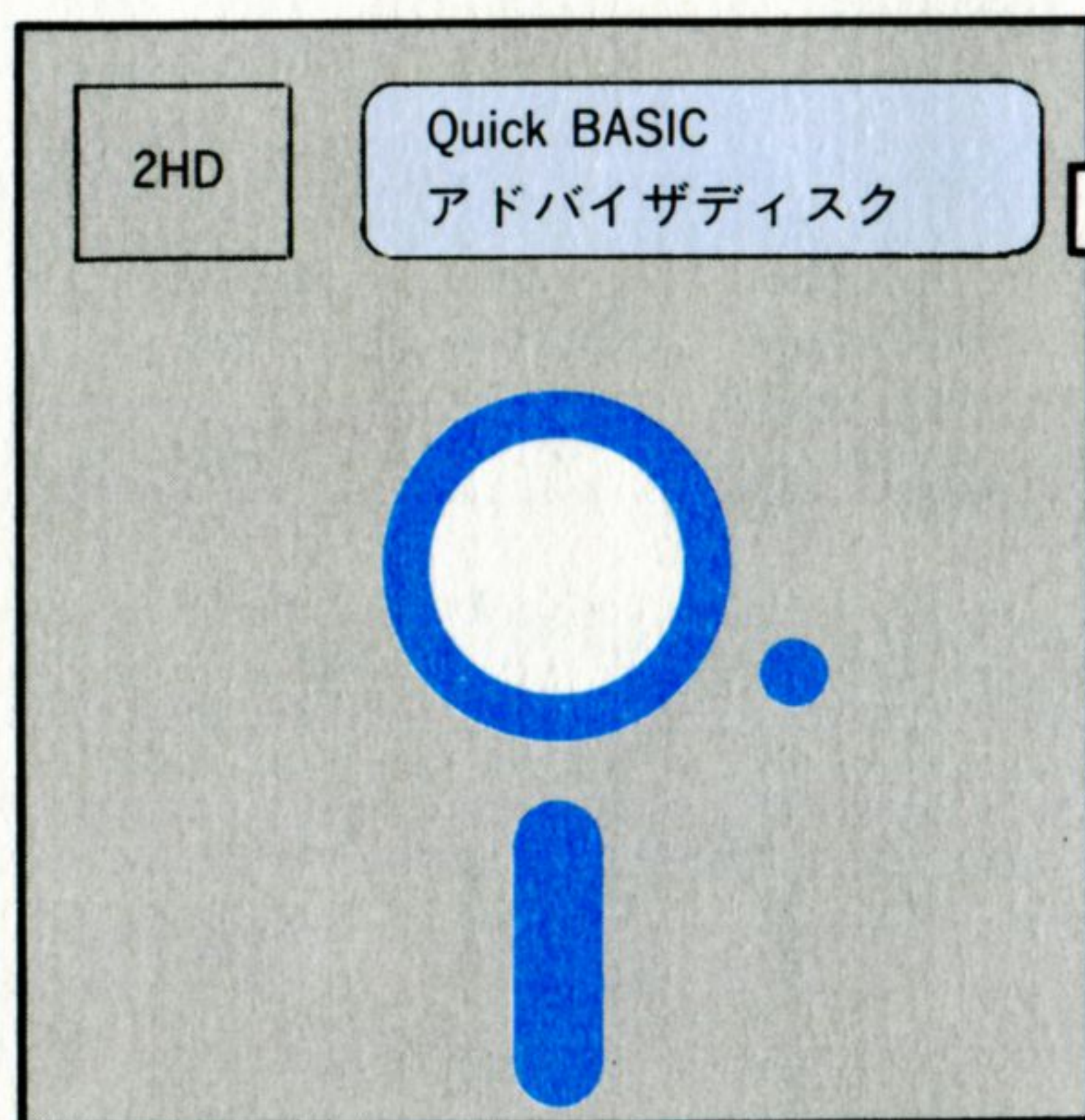
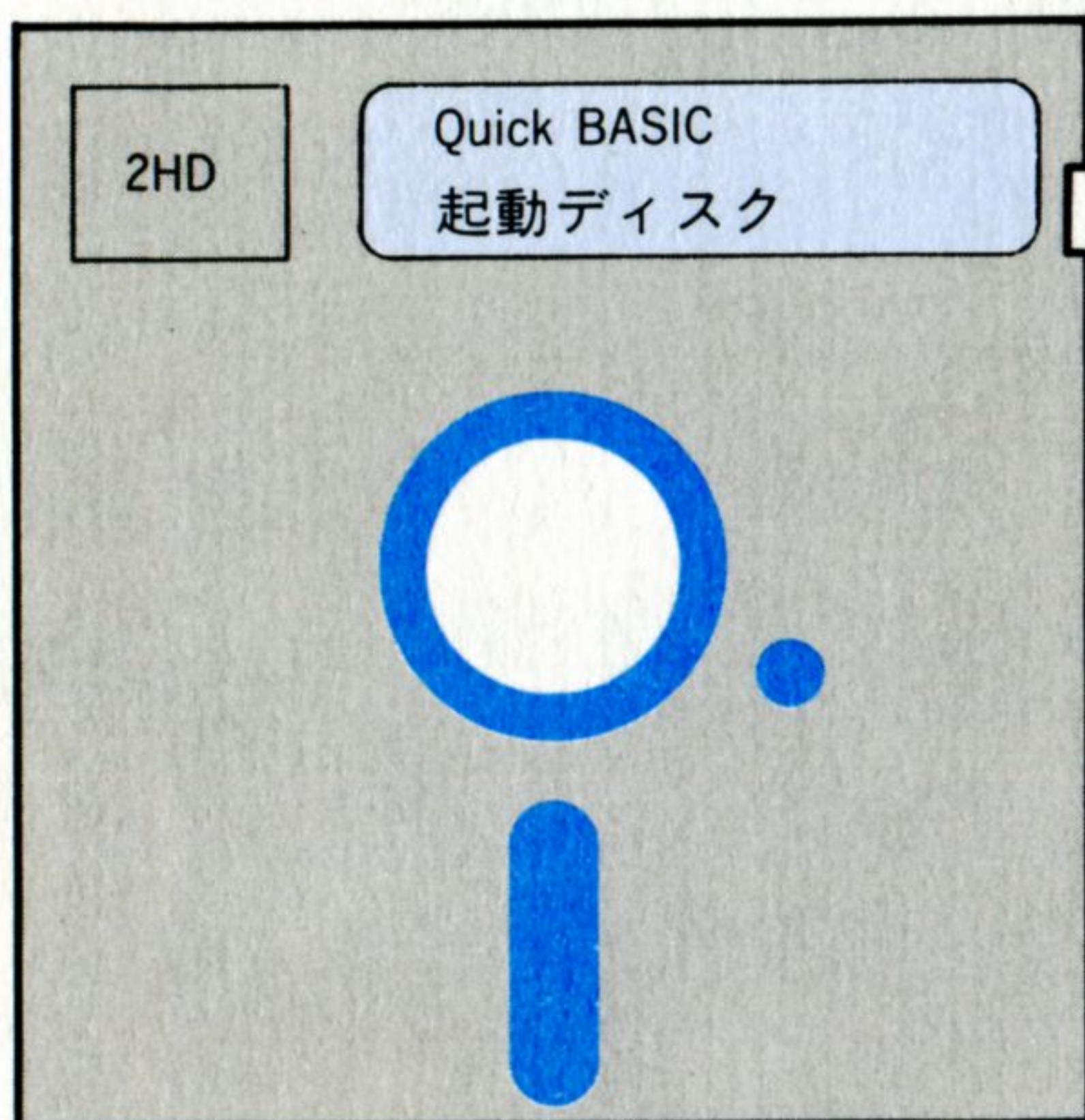
2HDの場合は、「QB起動ディスク」「QBアドバイザディスク」「QBワークディスク」の3枚のフロッピーディスクになります。

フォーマットのできたフロッピーディスクにはまちがえないように、

Quick BASIC　起動ディスク

Quick BASIC　アドバイザディスク

Quick BASIC　ワークディスク

と書いたシールを貼っておきます。

■フロッピーディスクのシール貼り

2DDのフロッピーディスクの場合は、起動ディスクとコンパイラディスクが別れるので、MS-DOSのシステムの入ったフロッピーディスクに、

Quick BASIC　起動ディスク

を貼ります。

残り3枚のフロッピーディスクには、

Quick BASIC　コンパイラディスク
Quick BASIC　アドバイザディスク
Quick BASIC　ワークディスク

と書いたシールを貼っておきます。

実行用ディスクの作成

つづいてAドライブ、Bドライブに入っているフロッピーディスクを取り替えて作業を進めます。

AドライブにQuick BASICのマスターディスクのセットアップディスクを入れます。

このとき、誤ってマスターディスクにデータなどを書き込んで、壊してしまわないように、すべてのマスターディスクのラベル横に切り込みがある場合は、プロテクトシールを貼っておきましょう。

Bドライブに先ほどフォーマットした「Quick BASIC起動ディスク」を入れます。

プロンプト「A>」で、キーボードから

SETUP⏎

と入力します。

すると、Quick BASICのセットアップ作業を行うセットアップユーティリティプログラムが起動されます。

Quick BASICのセットアップユーティリティからの注意が表示されますから、それを読みながら、何回か⏎キーを押すと、いよいよ質問に答えながら実行用ディスクをつくる作業に入ります。

マウスの使用を前提に日常使うディスクをつくる

最初に、画面にセットアップユーティリティのメニューが表示されます。

●メニュー選択してください・・・？1

ここでは、メニューの「1　Quick BASICの利用環境を作成します」を選択する意味で、初めから「1」が？マークのあとに設定されていて、この上でカーソルが点滅しています。

実行用フロッピーディスクを作成するのはメニューの「1」ですので、⏎キーを押して「1」を選択します。

画面が変わって、セットアップのための利用環境の設定に入ります。

●転送元のドライブ名を設定してください？　A

の質問には、「A」ドライブの意味で、初めから設定されている「A」の上でカーソルが点滅しています。

ここではマスターディスクがAドライブ、つまり1番のドライブに入っているので、そのまま⏎キーを押して「A」ドライブを選択する意思表示をします。

次に、

●どのドライブにインストールしますか？　B

実行用のQuick BASICを作成するフロッピーディスクが入っているドライブを質問されます。先ほどフォーマットしたディスクをBドライブ、2番のドライブに入れているので、このまま「B」を選択するために⏎キーを押します。

●インストール先はハードディスクですか［Y／N］？　Y

ここではハードディスクではなく、フロッピーディスクにインストールするので、「Y」ではなく、「No」の意味で「N」キーを押します。

インストールのドライブの関係の設定が終わったので、

●上記の設定でよろしいですか［Y／N］？　Y

と、確認の質問があります。「Yes」の意味で、初めから「Y」の上でカーソルが点滅しているので、まちがいがなければ⏎キーを押します。

まちがえていることに気がついた場合は、「N」キーを押して初めから設定をやり直します。

つづいてマウスやファイルの設定を行います。

●数値演算ライブラリを選択してください［1：Alt Math／2：Emula-tor Math］？　1

数値演算ライブラリ　シリアルマウス
実行用フロッピーディスク　マウス　マウスドライバ
インストール　バスマウス　ドキュメントファイル

この質問には、数値計算専用のコプロセッサを使用しているときは「2」を選択しますが、一般的には質問に設定されているとおり、「1」を選択しておくために、⏎キーを押します。

●マウスを使用しますか［Y／N］？　Y

マウスを持っている方は「Y」の意味で⏎キーを押します。持っていない方はキーボードから「N」を入力して⏎キーを押し、マウスを使用しないことを選択します。

Quick BASICは、マウスを利用しなくても使うことができますが、マウスを使ったほうがより快適に操作できます。できればマウスを使用したほうがいいでしょう。

ここでは、マウスを使用することを前提に進めます。

●マウスの種類を選択してください［1：バスマウス／2：シリアルマウス］？　1

使用するマウスの種類を選択します。

一般的に、パソコン本体のマウス用コネクタに接続されているマウスはバスマウスで、この場合は「1」の上で⏎キーを押します。

RS-232Cコネクタや、拡張用スロットに接続されることの多いシリアルマウスの場合は、キーボードから「2」を入力して⏎キーを押します。

画面の下半分に解説が出てくるので、それを参考に自分の使っているマウスの種類を「1」か「2」で選択します。

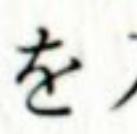
どちらのマウスかわからない場合は、マウスの取扱説明書を参照してください。

●マウスドライバを選択してください［1：MOUSE. SYS／2：MOUSE. COM］？　2

の質問は、マウスを使うときに起動するプログラムの指定です。

Quick BASIC以外のプログラムを使ったり、並行して他のプログラムを動かしたりするような高度な利用をしないかぎり、あまり気にする必要はありませんので、ここでは⏎キーを押しておきます。

●ドキュメントファイルを転送しますか［Y／N］？　Y

Quick BASICを利用するうえでの注意や、マニュアルの正誤表などの文書が

Quick BASICワークディスクに転送されます。ここでは、⏎キーを押して「Yes」の意思表示をします。

●サンプルプログラムを転送しますか［Y／N］？　Y

Quick BASICで作られたサンプルプログラムなどが、Quick BASICワークディスクに転送されます。QBアドバイザを使用中に、参照するように案内がありますので、⏎キーを押して「Yes」の意思表示をします。

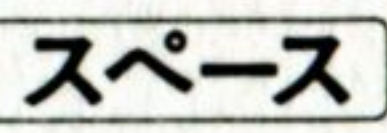
●上記の設定でよろしいですか［Y／N］？　Y

と、設定の確認を行います。

変更するところがなければ、そのまま⏎キーを押します。誤った設定をした場合は「N」を入力して⏎キーを押し、初めから設定をやり直します。

画面が変わって、これまで設定してきた条件で、フロッピーディスクにプログラムを登録する作業にかかります。

『マスターディスク』

●ドライブ『A』に『プログラムディスク』をセットして下さい。

『ユーザーディスク』

●ドライブ『B』に『QB起動ディスク』をセットして下さい。

【何かキーを押して下さい】

と、フロッピーディスクを差し替えるように、セットアップユーティリティから指示が出ます。

マスターディスクのプログラムディスクをドライブA、つまり「1」番のフロッピーディスクドライブに挿入します。ドライブB、「2」番のフロッピーディスクドライブには、先ほどフォーマットした実行用のフロッピーディスクのうち、「Quick BASIC　起動ディスク」を挿入します。

ディスクがきちんと挿入されていることを確認したら、スペースなどの適当なキーを押します。

セットアップユーティリティは、必要なプログラムを、自動的にマスターディスクから実行用のフロッピーディスクに複写していきます。

次つぎとセットアップユーティリティから、フロッピーディスクを差し替えるように指示が出るので、それに従ってマスターディスクと実行用ディスクを何度かまちがえないように入れ替えていきます。

Quick BASICを実行するために必要なプログラムを、すべて写し終えると、

読み込みが終了しました。

次に利用環境の作成を行ないます。
【何かのキーを押して下さい】
と指示が出て、キーを押すと、つづいて、
『ユーザーディスク』
●ドライブ『B』に『QB起動ディスク』をセットして下さい。
【何かキーを押して下さい】
と指示が出るので、初めに作成した「Quick BASIC　起動ディスク」にBドライブのフロッピーディスクを入れ替えて、キーを押します。すると、最後にMS-DOSの環境設定用のファイル「CONFIG. SYS」などをフロッピーディスクに作成して、実行用のディスクの作成が終了します。

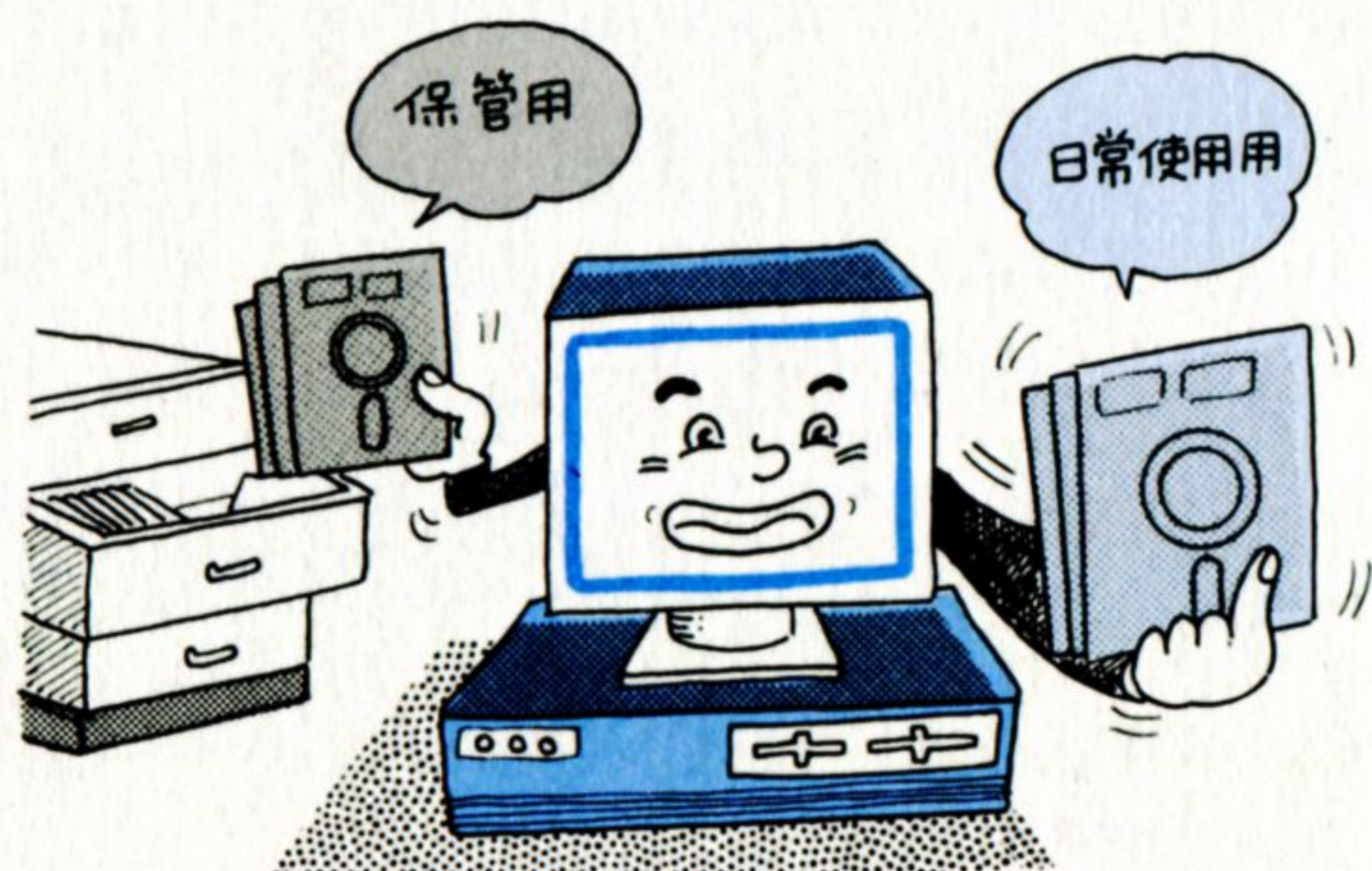
セットアップユーティリティの初めの画面に戻るので、「２」を選択して、Quick BASICの概要、操作方法の説明をひと通りおさらいしておきましょう。

セットアップユーティリティの終了は、初めの画面で「３」を選択します。

以後は、マスターディスクをたいせつに保管して、普段はここで作成した実行用のフロッピーディスクを使います。

最後に、Quick BASICの注意事項で、マニュアルなどに記載できなかった項目が記録されているファイルを読んでおきましょう。Aドライブに実行用のワークディスクを挿入して、「A>」のプロンプトで、

TYPE A：¥DOC¥README. DOC

と入力して⏎キーを押し、Quick BASICからのお知らせを読んでおきましょう。

日本語FEPの組み込み

日本語を入力するために

Quick BASICでは日本語を使うことができます。日本語入力を行うときには、「日本語FEP*」というプログラムが必要です。

Quick BASICでは、マイクロソフト社の標準的な日本語入力の方法である「MS-KANJI API」仕様に準拠した日本語FEPと、ジャストシステム社の代表的な日本語ワードプロセッサ「一太郎Ver.3」に組み込まれている日本語FEPを使用することができます。

日本語FEPは、単体またはソフトに組み込まれて発売されていますが、Quick BASICでは、

ATOK6	ジャストシステム
E1	イースト
Egbridge	エルゴソフト
VJE-β Ver.2	バックス

の日本語FEPを組み込んで利用することができます。

ここでは、「ATOK6」をQuick BASICに組み込んで利用することにしましょう。

日本語FEPに必要なファイルのコピー

ATOK6を利用するために、「一太郎」のシステムディスクを準備します。また、「一太郎」の辞書をシステムディスクと別のフロッピーディスクに登録している場合は、この辞書ディスクも準備します。

ATOK6の利用に必要なプログラムファイルは、

ATOK6A. SYS
ATOK6B. SYS
ATOK. DIC

の3つのファイルです。

Quick BASICのセットアップに引きつづいて、日本語FEPを組み込む作業をしましょう。

ATOK6A. SYS
ATOK6B. SYS
MS-KANJI API　　ATOK. DIC

Aドライブに ATOK6 の入った「一太郎」のシステムディスクを入れます。

Bドライブに、いま作成した「Quick BASIC　起動ディスク」を入れます。

準備ができたら、MS-DOS のコマンド待ちの状態である「A>」のプロンプトにつづいて、

COPY　A：ATOK6A. SYS　B：⏎

とキーボードから入力します。Aドライブの「一太郎」のシステムディスクから、BドライブのQuick BASIC の起動ディスクに「ATOK6A. SYS」のファイルがコピーされます。つづいて、

COPY　A：ATOK6B. SYS　B：⏎

と、入力します。同じように起動ディスクに「ATOK 6B. SYS」のファイルがコピーされます。

次に、ATOK6 の辞書である「ATOK. DIC」のファイルをコピーします。

Aドライブに、ATOK6 の辞書の入ったフロッピーディスクを入れます。

Bドライブに、「Quick BASIC　ワークディスク」を入れます。

準備ができたら、MS-DOS のコマンド待ちの状態である「A>」のプロンプトにつづいて、

COPY　A：ATOK. DIC　B：⏎

と、入力します。

AドライブからBドライブへ「ATOK. DIC」のファイルがコピーされます。

これで、日本語 FEP を使うために必要なプログラムファイルが、Quick BASIC の起動ディスクとワークディスクにコピーされました。

※日本語 FEP（日本語フロントエンドプロセッサ）　パソコンソフトに対して、日本語の入力を受け持つ専用のソフト。

パソコンソフトは、BASIC 言語やデータベースなどの本来のしごと専用につくられています。

日本語の入力は英語などと違って、漢字を使うためキーボード上にすべての文字を配置することができません。そこで、カナやローマ字から漢字に変換する「かな漢字変換」の機能をもっていないと、日本語を使用することができません。

日本語 FEP は、パソコンソフトより前にキーボードからの入力を受け付けて、かな漢字変換を行い、その結果を日本語のデータとして、パソコンソフトに引き渡すしごとをしています。

CONFIG. SYSファイルの書き換え

プログラムファイルをディスクに登録するだけでは、日本語FEPは動きません。日本語FEPを動かすためには、パソコンの動作を管理しているMS-DOSに「日本語FEP用のプログラムを使います」という設定を与えなければなりません。その設定が書き込まれているのが「CONFIG. SYS」というファイルです。

このファイルを書き換えるには、エディタやワープロのソフトが必要になります。ここでは、「一太郎」を使って書き換えてみましょう。

まず、Aドライブ、Bドライブに入っているフロッピーディスクを、「一太郎」のシステムフロッピーディスクに差し替えて、「一太郎」を起動します。

Bドライブを「Quick BASIC　起動ディスク」に差し替えます。

「一太郎」の「T（フ）ファイル・L（ヨ）読込み」のコマンドを使って、

B：¥CONFIG. SYS

のファイル名を指定して、画面上に「CONFIG. SYS」のファイルを呼び出します。

画面には「CONFIG. SYS」ファイルの内容が、次のように表示されています。

一部、違うように表示されている場合もあるかもしれませんが、それはそのままにしておきます。

```
FILES=20
BUFFERS=20
```

この2行のあとにキーボードから、次の2行を書き加えます。

```
device=atok6a. sys　/d=b：¥atok.dic　/s=1　/e=1　/t=1
/b=0
device=atok6b. sys
```

ここでは追加したことがわかるように小文字で書きましたが、大文字でもかまいません。画面は、上の2行が追加されて、次のようになっているはずです。

```
FILES=20
BUFFERS=20
device=atok6a. sys　/d=b：¥atok.dic　/s=1　/e=1　/t=1
/b=0
```

T（フ）ファイル・S（ホ）保存
CONFIG. SYS
T（フ）ファイル・L（ヨ）読み込み　　日本語 FEP

device＝atok6b. sys

このファイルを「一太郎」の「T（フ）ファイル・S（ホ）保存」のコマンドを使って、Bドライブに書き戻します。

ファイル名が「B：¥CONFIG. SYS」になっているかどうか確認してください。違ってしまっているときは指定し直します。

これで、日本語FEPの組み込みが終わりました。

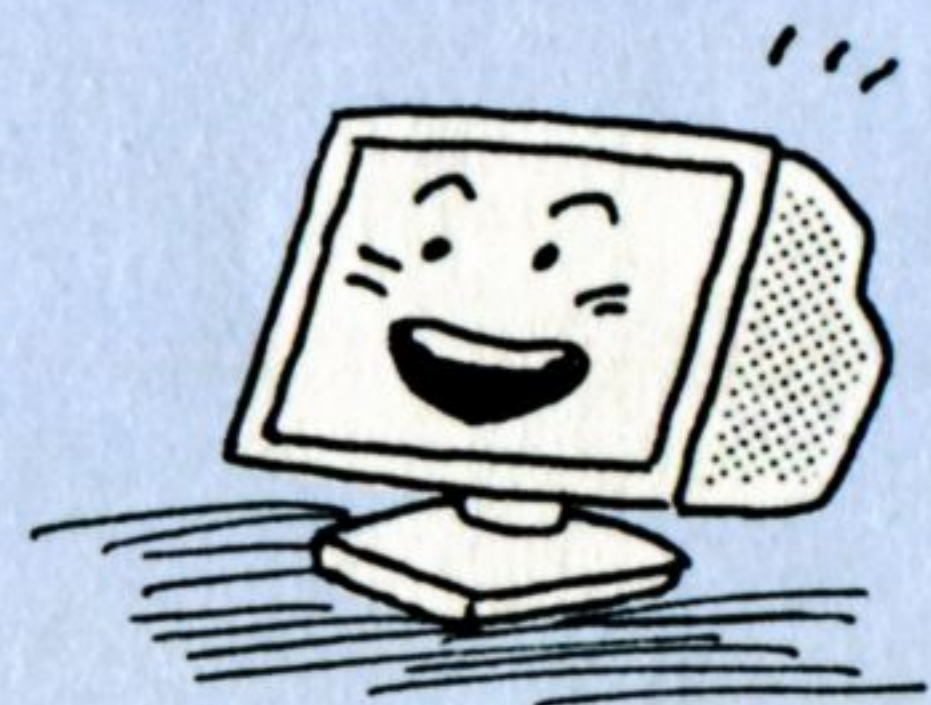
次からは、Aドライブに、2HDの場合「Quick BASIC　起動ディスク」を、Bドライブに「Quick BASIC　ワークディスク」を入れて、本体の電源スイッチをONにするか、電源スイッチが入っている場合は、リセットスイッチを押すと、Quick BASICが起動すると同時に、日本語FEPが使えるようになります。

起動ディスク／ワークディスクの内容

ここまでの作業で作成された「Quick BASIC　起動ディスク」、「Quick BASIC　ワークディスク」の内容を示します。

セットアップのときの指定によって一部違ってくる場合もありますが、参考にしてください（2HDフロッピーディスクの場合）。

■「Quick BASIC　起動ディスク」ファイルリスト

```
  ドライブ A: のディスクのボリュームラベルはありません.
  ディレクトリは A:¥

COMMAND  COM     17190  83-09-19  16:19
AUTOEXEC BAT        73  90-01-01   8:59
CONFIG   SYS       126  90-01-01   8:59
ATOK6A   SYS     54353  88-08-11  12:00
ATOK6B   SYS     21228  88-08-11  12:00
BIN          <DIR>      90-01-01  17:17
LIB          <DIR>      90-01-01  17:17
        7 個のファイルがあります.
    98304 バイトが使用可能です.
```

■起動ディスク内の「ディレクトリBIN」ファイルリスト

```
 ドライブ A: のディスクにはボリュームラベルがありません
 ディレクトリは A:¥BIN

.             <DIR>      90-01-01  17:17
..            <DIR>      90-01-01  17:17
QB       EXE    311758   89-11-15   4:50
QB       INI        77   90-01-01  17:17
BRUN45A  EXE     92329   89-11-15   4:50
LINK     EXE     69133   88-09-07  16:27
LIB      EXE     35643   88-07-26  10:52
BC       EXE    111479   89-11-15   4:50
MOUSE    COM      8431   89-10-18   2:00
        9 個のファイルがあります
    98304 バイトが使用可能です
```

■起動ディスク内の「ディレクトリLIB」ファイルリスト

```
 ドライブ A: のディスクにはボリュームラベルがありません
 ディレクトリは A:¥LIB

.             <DIR>      90-01-01  17:17
..            <DIR>      90-01-01  17:17
BQLB45   LIB     25301   89-11-15   4:50
QB       LIB      2075   89-11-15   4:50
QB       QLB      5814   89-11-15   4:50
BCOM45A  LIB    239343   89-11-15   4:50
BRUN45A  LIB     25769   89-11-15   4:50
GEN      LIB     13871   89-11-15   4:50
GEN      QLB     13843   89-11-15   4:50
GRAPH    LIB      6181   89-11-15   4:50
GRAPH    QLB      9295   89-11-15   4:50
MOUSE    LIB      5675   89-11-15   4:50
MOUSE    QLB      8083   89-11-15   4:50
       13 個のファイルがあります
    98304 バイトが使用可能です
```

■「AUTOEXEC.BAT」ファイルの内容（例）

```
PATH=A:¥BIN
SET LIB=A:¥LIB
SET INCLUDE=B:¥INCLUDE
SET TMP=B:¥
MOUSE
```

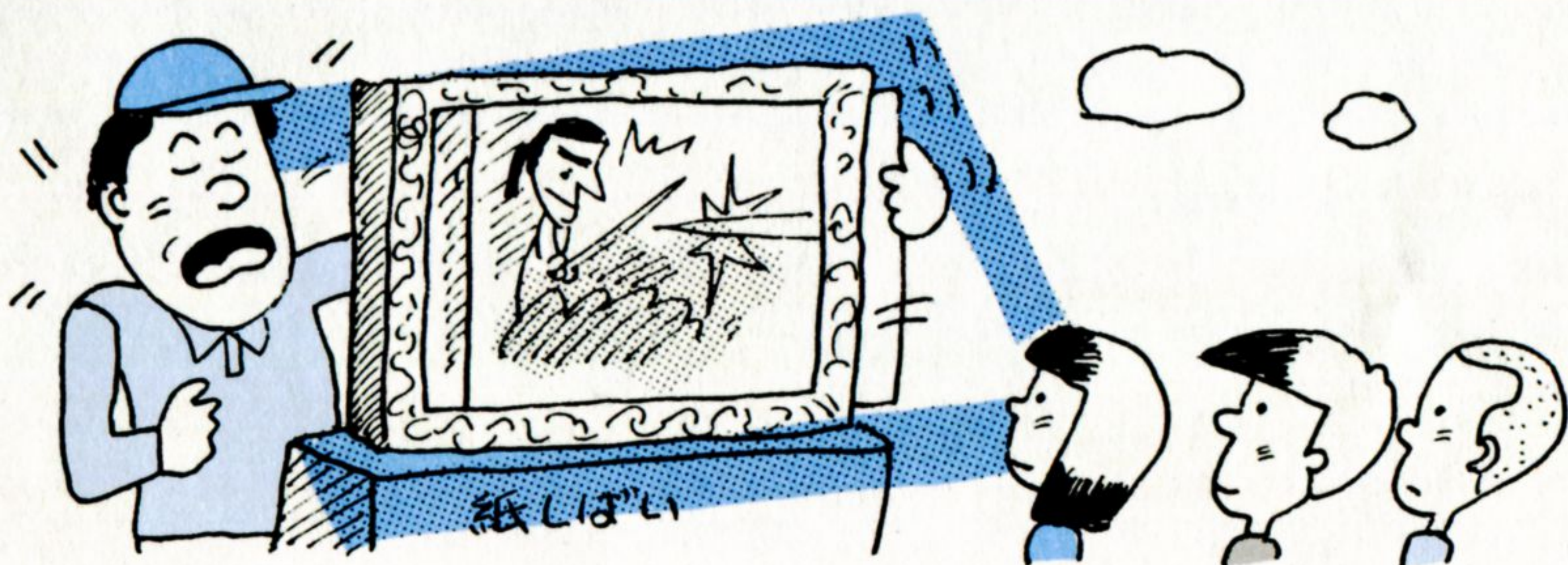
■「CONFIG.SYS」ファイルの内容（例）

```
SHELL=A:¥COMMAND.COM A:¥ /P
FILES=20
BUFFERS=10
device=atok6a.sys /d=b:¥atok.dic /s=1 /e=1 /t=1 /b=0
device=atok6b.sys
```

■「Quick BASIC　ワークディスク」ファイルリスト

```
ドライブ B: のディスクのボリュームラベルはありません.
ディレクトリは B:¥

INCLUDE          <DIR>        89-12-29     8:49
SOURCE           <DIR>        89-12-29     8:49
ADVR             <DIR>        89-12-29     8:53
DOC              <DIR>        89-12-29     8:55
ATOK       DIC     445952     88-08-11    12:00
BAS              <DIR>        90-01-01    17:19
          6 個のファイルがあります.
     161792 バイトが使用可能です.
```

■ワークディスク内の「ディレクトリ BAS」ファイルリスト（例）

```
ドライブ B: のディスクにはボリュームラベルがありません
ディレクトリは B:¥BAS

.                <DIR>        90-01-01     6:05
..               <DIR>        90-01-01     6:05
GOAISATU BAS          559     90-01-01     7:32
TAIJUU   BAS         1320     90-01-02    18:12
SUUJIATE BAS         1817     90-01-03    13:44
SHUKEI   BAS         1974     90-01-04    20:02
MENU     BAS         1761     90-01-05    10:43
YESNO    BAS         2393     90-01-06     9:12
JIKAN    BAS         1318     90-01-07    22:15
RINGO    BAS         1667     90-01-08    18:29
HOSIZORA BAS          493     90-01-08    19:58
BOGRAPH  BAS         1146     90-01-09     7:57
ENGRAPH  BAS         1416     90-01-09    11:51
DENWA    BAS         7795     90-01-10    12:15
DENWA    DAT          293     90-01-10    20:49
MEISISTR BAS         9796     90-01-12    15:36
MEISISTR DAT         1890     90-01-12    22:19
        17 個のファイルがあります
    812032 バイトが使用可能です
```

初めと終わり

始めたけど終われないではだめだから

さあ、前につくったシステムディスクを使って、Quick BASICを起動してみましょう。

まずは、いったん始めてみたものの終わりをどうしていいかわからなくなった場合のことを考えて、ここではQuick BASICの終わり方もいっしょに説明しておきます。

Quick BASICの起動

●フロッピーディスクの挿入

本体や周辺機器の接続が正しく行われているかどうかを確認してください。

ここからは、次の構成を前提として説明します。

2HDのフロッピーディスクドライブが2台

カラーディスプレイ

マウス

プリンタ

ハードディスクやRAMディスクが接続してある場合は、ドライブ名が本書での説明と違う場合がありますので、各自の構成に合わせて適切なドライブ名に読み替えてください。

それでは、いよいよスタートです。

Aドライブには「Quick BASIC　起動ディスク」、Bドライブに「Quick BASIC　ワークディスク」を、それぞれ方向をまちがえないように入れます。

本体のスイッチをオンにします。つづけて、素早くフロッピーディスクについているレバーをカチッと止まるまで、時計方向に回します。

● MS-DOSの起動

まず、MS-DOSが起動します。

各自の設定によっては、日付や時間の確認などをしてくる場合があります。そのときは、⏎キーを押すなどして、コマンド待ちの状態にしてください。

MS-DOSのコマンド入力待ちは、A>のプロンプトが表示されている状態

です。

ディスプレイの画面にこのプロンプトが表示されましたか？　これで、MS-DOS が起動しました。

A>

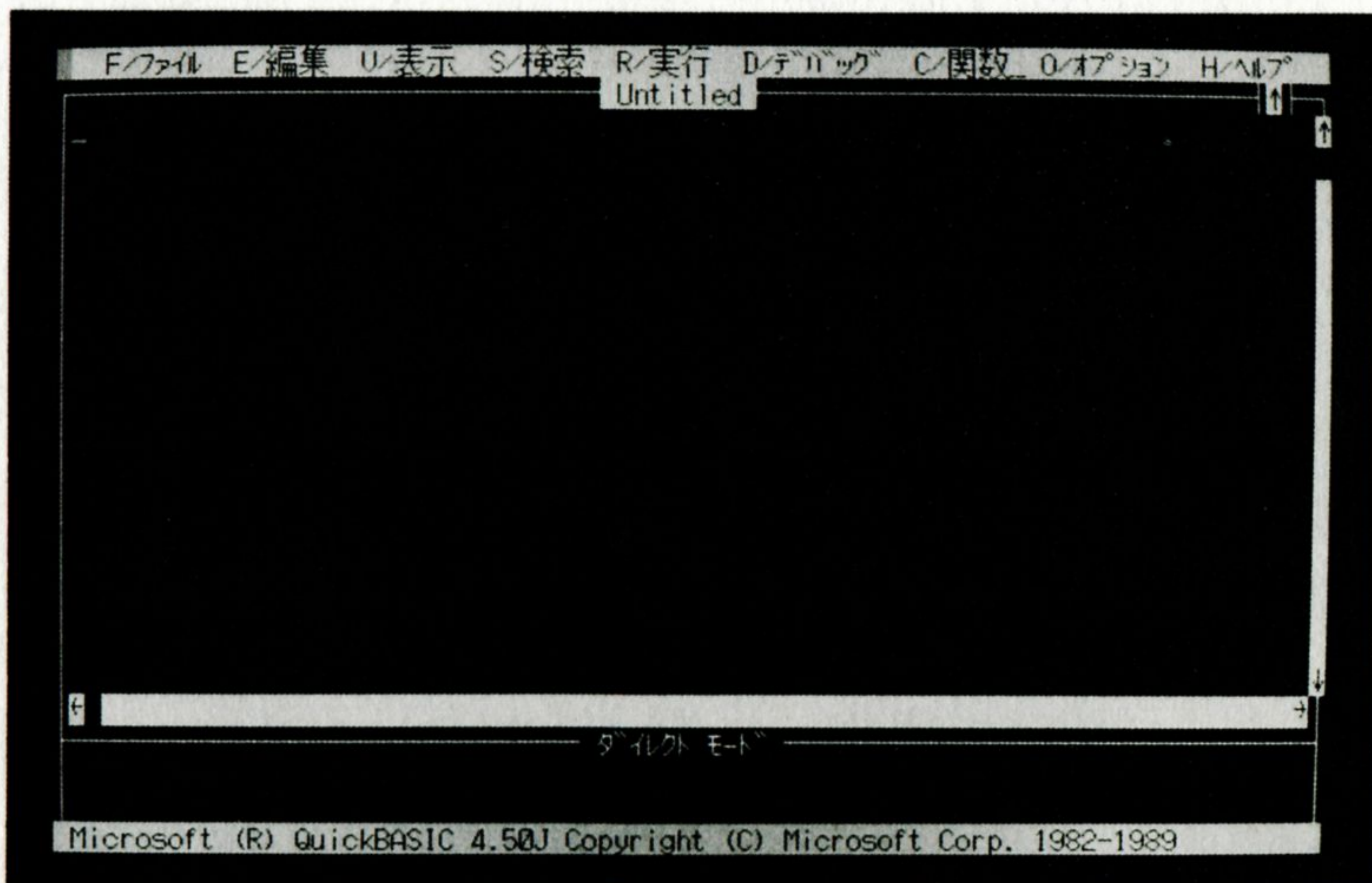
■コマンド待ち

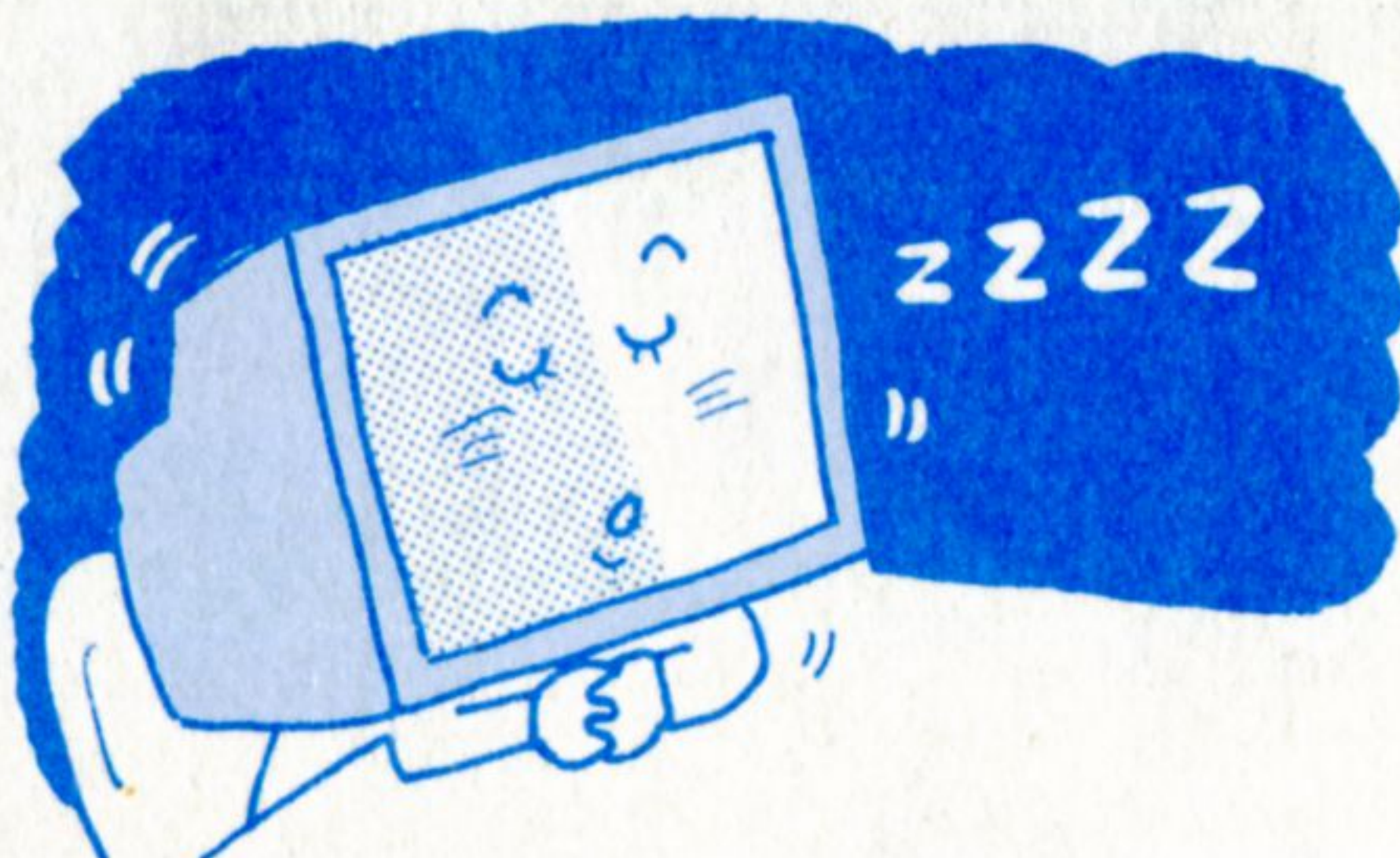
● Quick BASIC の起動

A>のプロンプトにつづいて、キーボードから、

QB ⏎

と入力します。

すると、カチャカチャとフロッピーディスクにアクセスする音が聞こえて、ディスプレイに、次のような Quick BASIC の初期画面が表示されます。

■Quick BASIC の初期画面

どうしても、表示されない場合は、フロッピーディスクの挿入してあるドライブにまちがいがないかどうか確かめてください。それでも起動できない場合は、41ページから正確にやり直してください。

Quick BASICの終了

●コマンドの選択

終了のコマンドを選択して、Quick BASICを終了します。

画面のいちばん上のメニューバーのなかから「メニュー選択」して、プルダウンメニューを開き、コマンドを選択します。

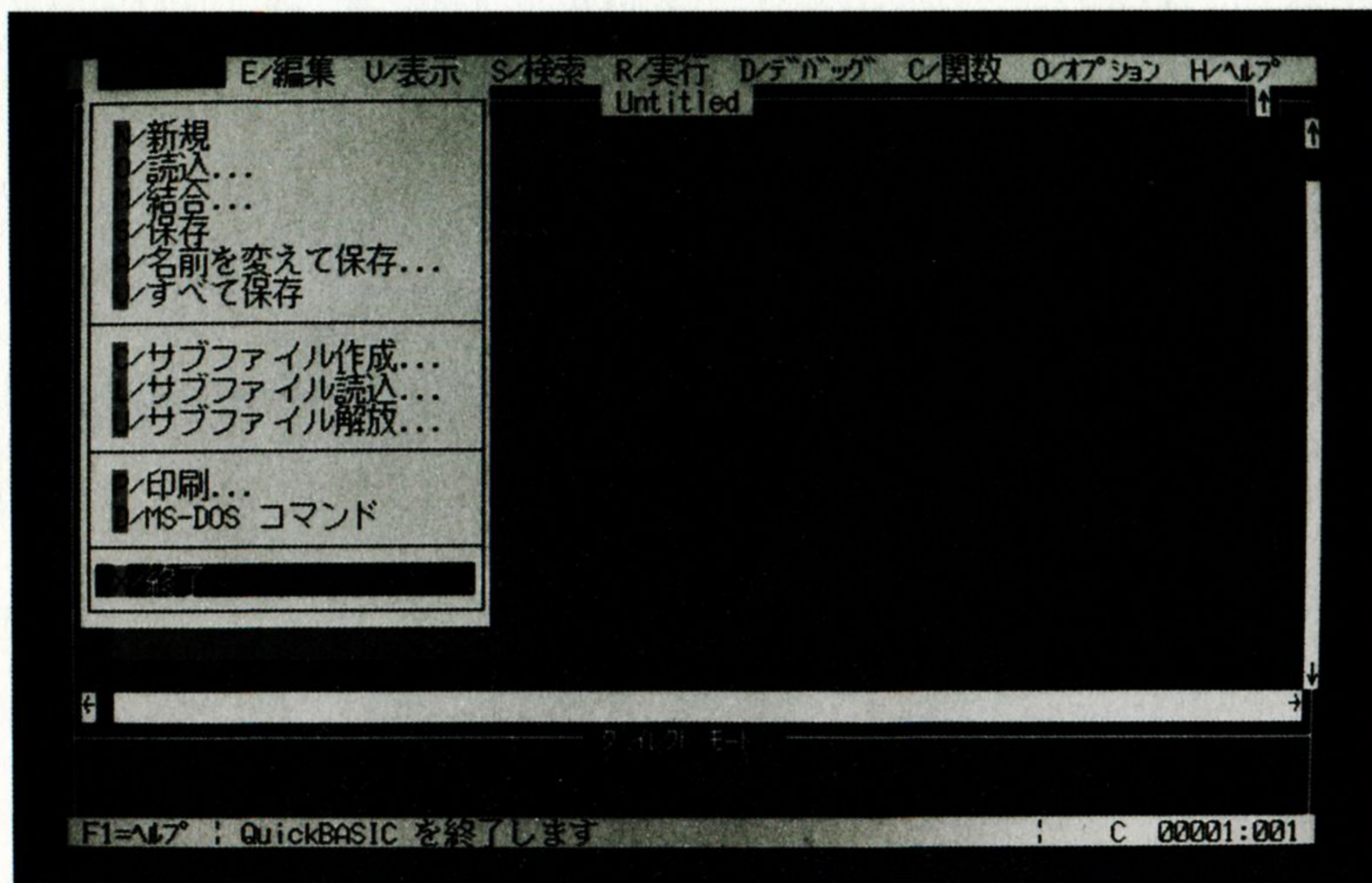
そのなかから目的のコマンドを「コマンド選択」するという手順です。

ここでは、とりあえず「終了」のコマンドを選択して、実行してみましょう。

(1)マウスを使用している場合

マウスの場合は、メニューバーのF／ファイルにマウスカーソルを合わせて左クリックします。

プルダウンメニューが開かれるので、このなかのX／終了にマウスカーソルを合わせて、左クリックします。

これで、終了のコマンドが選択されます。

■プルダウンメニューの画面

(2)キー入力の場合

まず、メニューバーの選択モードに入ります。GRPHキーを押すと、メニューバーのなかで現在選択されているメニューがアクティブとなって、黒地に黄色文字で反転表示されます。

このアクティブメニューをカーソルキーで移動して、F／ファイルをアクテ

メニュー選択	F／ファイル
コマンド選択	X／終了
プルダウンメニュー	エンベロープ

ィブにして、↵キーを押して選択するか、またはキーボードの「F」キーを押して、プルダウンメニューを開きます。

メニューのなかから同じように、カーソルで、X／終了のコマンドを選択して↵キーを押すか、またはキーボードの「X」キーを押します。

少し待つと、Quick BASICが終了して、MS-DOSのコマンド待ちの状態に戻ります。

このようにQuick BASICでは、メニュー選択からのコマンド選択で、いろいろな操作ができます。

キーボードからも、またマウスを使っても操作することができます。

●終了

Quick BASICが終了すると、MS-DOSのコマンド待ちの状態に戻って、A >のプロンプトが、ディスプレイの画面に表示されています。

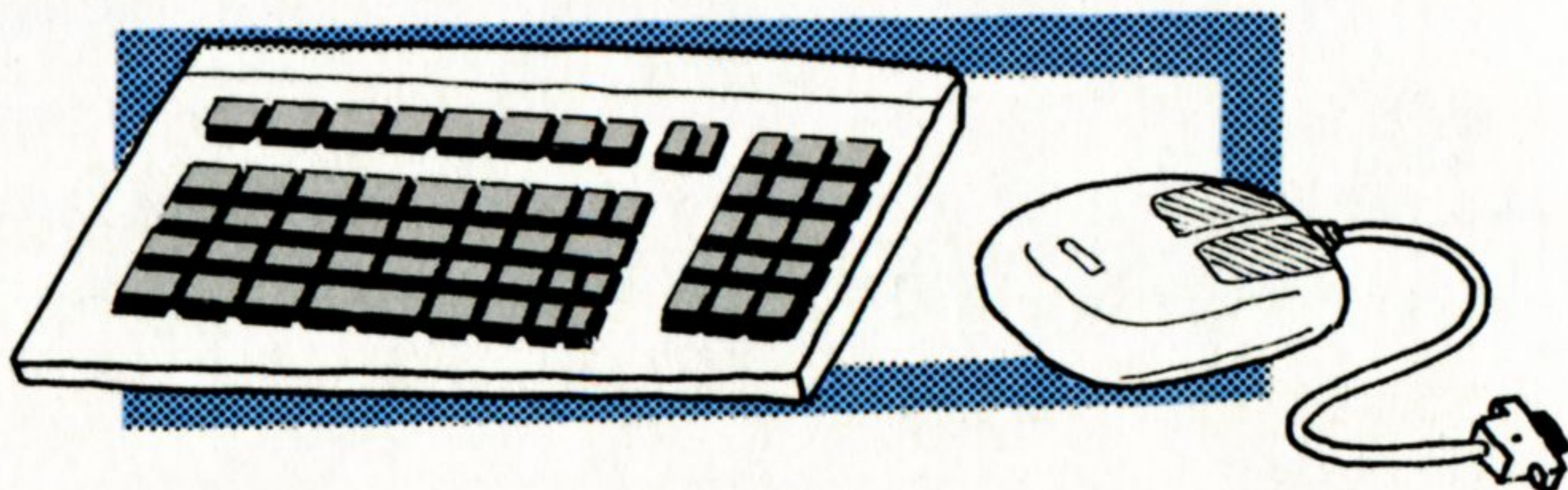
フロッピーディスクのレバーを時計方向と反対に回して、ドライブからA、Bのフロッピーディスクを抜きます。

ドライブから出したフロッピーディスクは、傷や汚れがつかないように、すぐにエンベロープ（フロッピーディスク収納用の紙袋）に入れて、しまっておきます。

ディスクドライブが空になったら、本体のスイッチをオフにします。

これで、Quick BASICの初めから、終わりまでをひと通り試してみました。

このようにQuick BASICでは、簡単にコマンドの選択、入力などが行えるようになっているのがわかったと思います。

※キーボードやマウスを使っての操作方法の詳細は、PART 2　75ページからの「Quick BASICとの対話」を参照してください。

How To プログラミング

エディタ画面を使って日本語も入力できる

プログラムの入力

Quick BASICを起動すると、エディタモードの画面からスタートします。Quick BASICのプログラミングは、このエディタを使って行います。

実際に簡単なプログラムをつくってみましょう。ディスプレイ画面のいちばん上にはメニューバー、その下のタイトルバーには「Untitled」と表示されていて、まだプログラムの名まえがつけられていないことを示しています。

カーソルはエディタ画面の左上で点滅しています。この位置からプログラムを入力していきます。

キーボードから、

PRINT"

と、入力してください。

次に、日本語の文字を入力するために、日本語入力モードに入ります。

[CTRL]+[XFER]（2つのキーを同時に押す）とすると、画面のいちばん下の行が日本語入力ラインに変わって、組み込まれた日本語FEPが起動します。

Quick BASICのインストールで組み込んだATOK6の場合は、「－」記号のカーソルが現れます。

FEPの機能を利用して、

こんにちわ[↵]

と、キーボードから入力して、日本語を確定します。

次に、確定した日本語の文字列をプログラムのなかに入れます。

ここで[↵]キーを押すと、エディタ画面のカーソルの位置にいま入力した文字が表示されて、プログラムの一部になったことがわかります。

日本語入力が終わりましたので、日本語FEPの機能を終了します。再度、[CTRL]+[XFER]とすると、日本語入力モードから通常の入力モードに戻って、画面のいちばん下の日本語入力ラインが消え、組み込まれた日本語FEPが終了したことがわかります。

エディタ画面のカーソル位置に戻って、⏎キーを押して、プログラムの1行目の入力を終わって、改行します。

つづけて、次のようにプログラムを3行入力しましょう。

INPUT　A⏎

INPUT　B⏎

PTINT　A+B⏎

おや、3行目の「PTINT」がおかしいですね。でもここでは、このままにしておきましょう。これで、非常に簡単なプログラムがひとつできました。

プログラムの修正

このプログラムはまちがいがあります。最後の行の「PTINT」は、正しくは「PRINT」です。

プログラムを修正してみましょう。

⬆⬇⬅➡のカーソルキーを使って、PTINTの「I」の位置にカーソルを合わせます。

BSを押して、ひとつ前の文字「T」を消します。

Rと入力すると、「PRINT」に訂正されました。

これで、正しいプログラムになりました。

プログラムの修正や訂正も簡単にできます。また、編集メニューを使えば、文字列のコピーや移動などの編集作業も、ワープロ感覚で快適にすることができます。

プログラムの実行は

コマンドはファンクションキーに登録されている

プログラムの実行

プログラムがひとつできあがったので、さっそく実行してみましょう。

●メニューからのコマンド選択

マウスまたは、キーボードからのメニュー選択では、メニューバーから「R／実行」を選択してプルダウンメニューを開き、「S／スタート」のコマンドを選択します。

このように、マウス、またはキーボードを使ってメニューのなかからコマンドを選択していくのは、初めはわかりやすいのですが、操作に慣れてくるとめんどうになってきます。

そこで、Quick BASICでは、よく使われるコマンドはファンクションキーにショートカットキーとして登録されていて、そのショートカットキーをひと押しするだけで、コマンドを実行することができるようになっています。

●ファンクションキーでのコマンド入力

Quick BASICに付属してくるテンプレートをファンクションキーの上に乗せると、どのキーがどのコマンドに対応しているかがわかります。

プログラムの「実行」コマンドは、ファンクションキーの SHIFT ＋ f・5 に登録されています。

SHIFT キーを押しながらファンクションキー f・5 を押してみましょう。

いままでのエディタ画面から、実行中の画面に切り替わりました。

プログラムの1行目を実行して、「こんにちは」と表示され、つづいて次の行に「？」とクエスチョンマークが表示されて、2行目の「INPUT　A」で、Aに代入される数値の入力を求めています。

ここでは、例えば、3 ⏎ とキーボードから入力します。

つづいて次の行に「？」とクエスチョンマークが表示されて、3行目の「INPUT　B」で、Bに代入される数値の入力を求めています。

同じように、例えば、6 ⏎ とすると、画面には4行目の「PRINT　A+B」

の実行結果である、「9」が表示されます。

画面下には、

何かキーを押してください。

と表示されています。

これは、Quick BASICがプログラムの実行を終わったので、キーを押せばエディタ画面に戻りますよと、案内しているのです。

［スペース］バーを押してみましょう。画面がエディタ画面に切り替わって、いま実行したプログラムが表示されています。

Quick BASICでは、画面が切り替わってプログラムを実行するので、プログラムのエディットと実行がはっきり分かれていて、わかりやすくなっています。

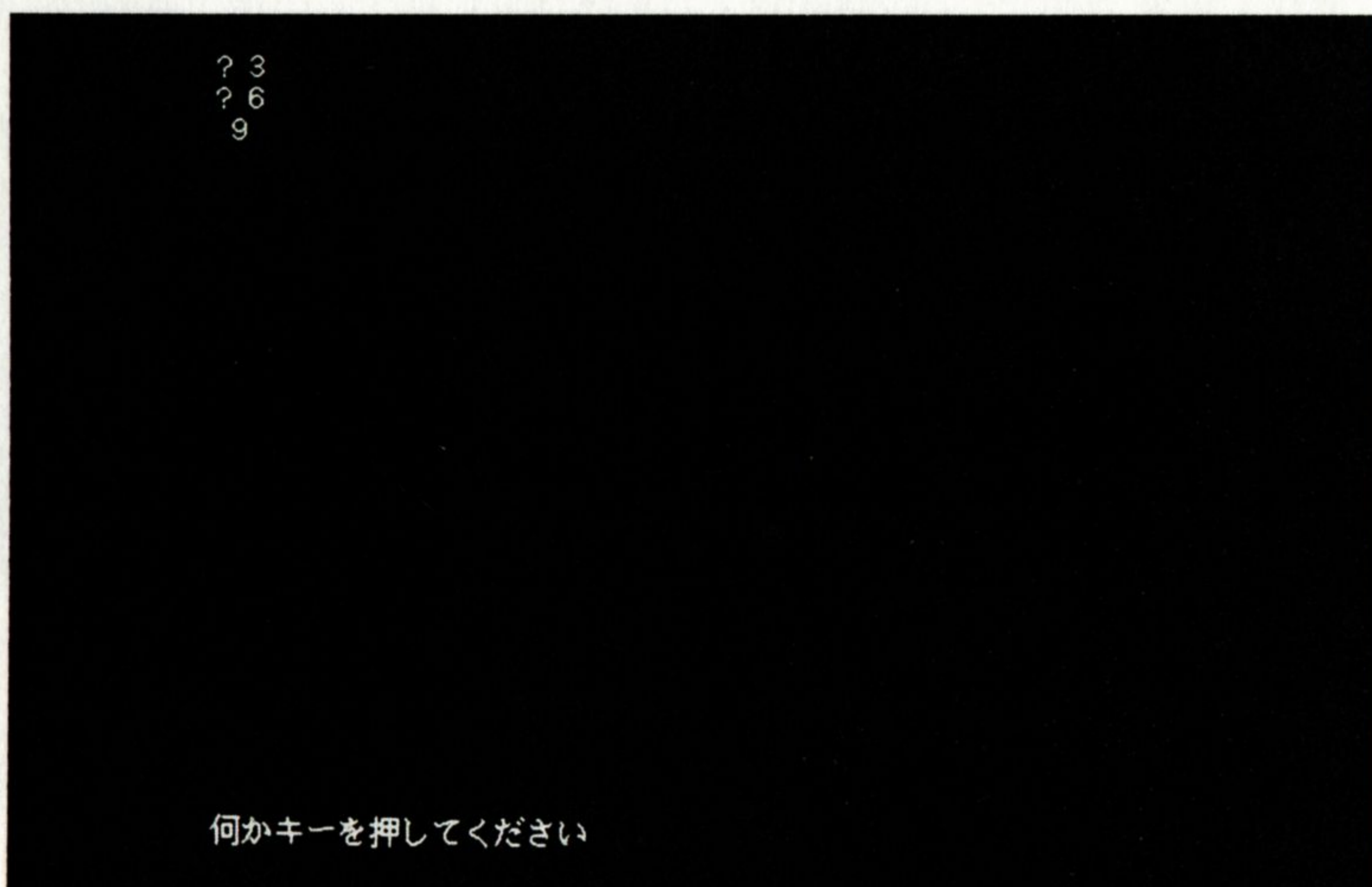
一時停止

プログラムを実行しているときに、プログラムを変更する必要があったりして、途中で実行を一時中断したい場合があります。

プログラムの一時停止は、［STOP］キーを押して行います。

それでは試してみましょう。

```
? 3
? 6
 9




何かキーを押してください
```

プログラムの実行画面

ファンクションキー［f・5］を押して、プログラムを実行させてください。

実行画面に、「こんにちは」と表示され、つづいて次の行に「？」とクエスチョンマークが表示されます。

キーボードから3［↵］と入力します。

ここで、［STOP］を押すと、画面が切り替わって、エディタ画面に移ります。

画面では、実行中のプログラムの3行目の「INPUT　B」の行が緑色に変化していて、この行を実行中にプログラムがストップしたことを示しています。

Quick BASICではこのように、プログラムの実行が一時停止されてエディタ画面に戻ると、次に実行される行を緑色で示して、カーソルを合わせます。この行に手を加えたい場合のために、非常に便利です。

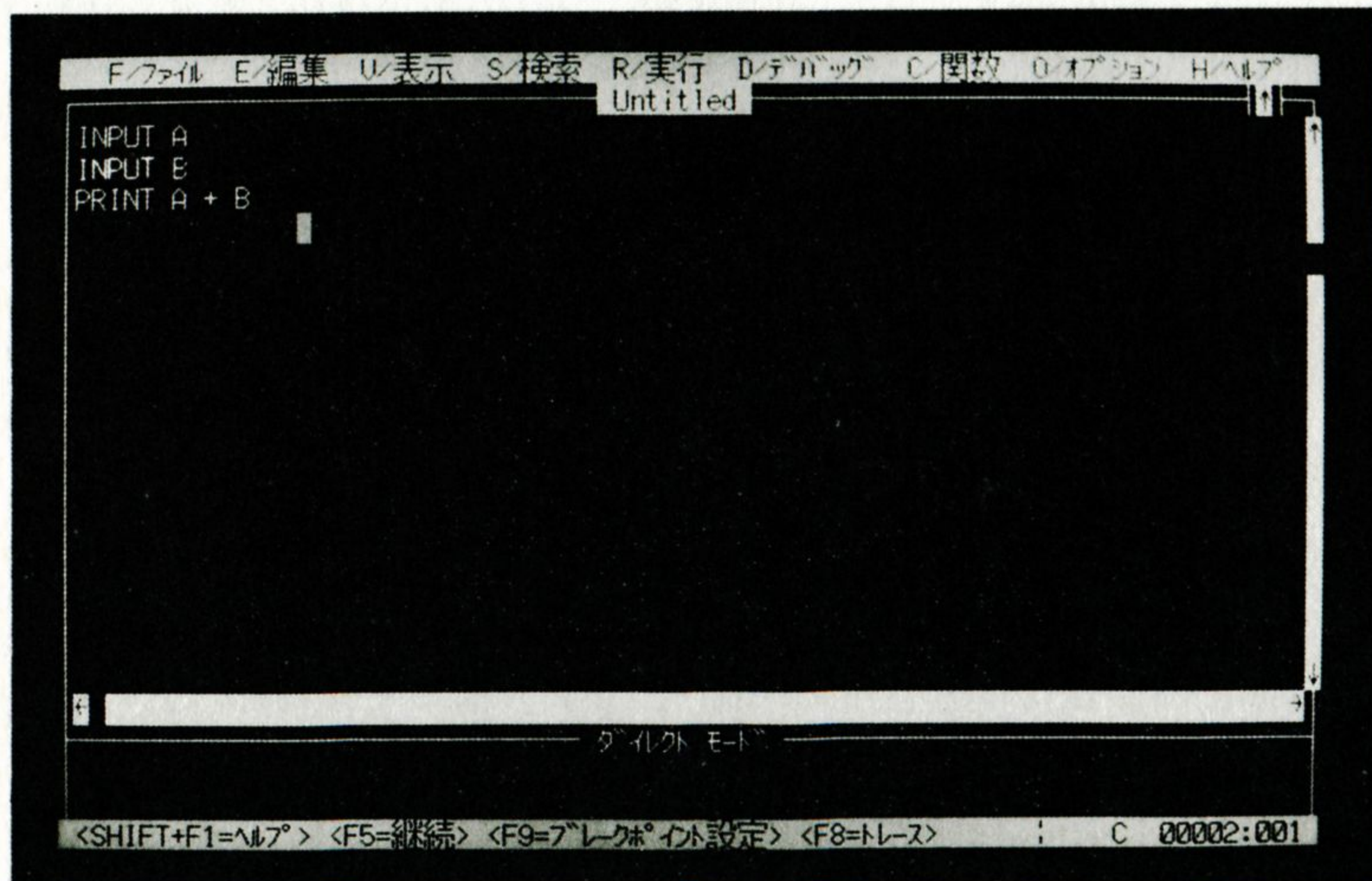
再スタート

プログラムの一時停止を解除して、つづけて実行してみます。プログラムの続行もファンクションキーに割り当てられています。

［f・5］を押すと、実行画面に戻って、再度「？」と、Bに数値の代入を求めてきます。

6［↵］とキーボードから入力すると、実行結果の「9」が表示されて、画面下に、

何かキーを押してください。

```
F/ファイル E/編集 V/表示 S/検索 R/実行 D/デバッグ C/関数 O/オプション H/ヘルプ
Untitled
INPUT A
INPUT B
PRINT A + B

ダイレクト モード
<SHIFT+F1=ヘルプ> <F5=継続> <F9=ブレークポイント設定> <F8=トレース>   C 00002:001
```

一時停止の画面

と表示され、プログラムの実行が終わったことがわかります。

[スペース]を押すと、エディタ画面に戻ります。

このようにQuick BASICでは、プログラムの実行と、一時停止、エディタ画面でのプログラムの修正をつづければ、効率的にプログラムのデバッグ（まちがいの訂正）を行えるようになっています。

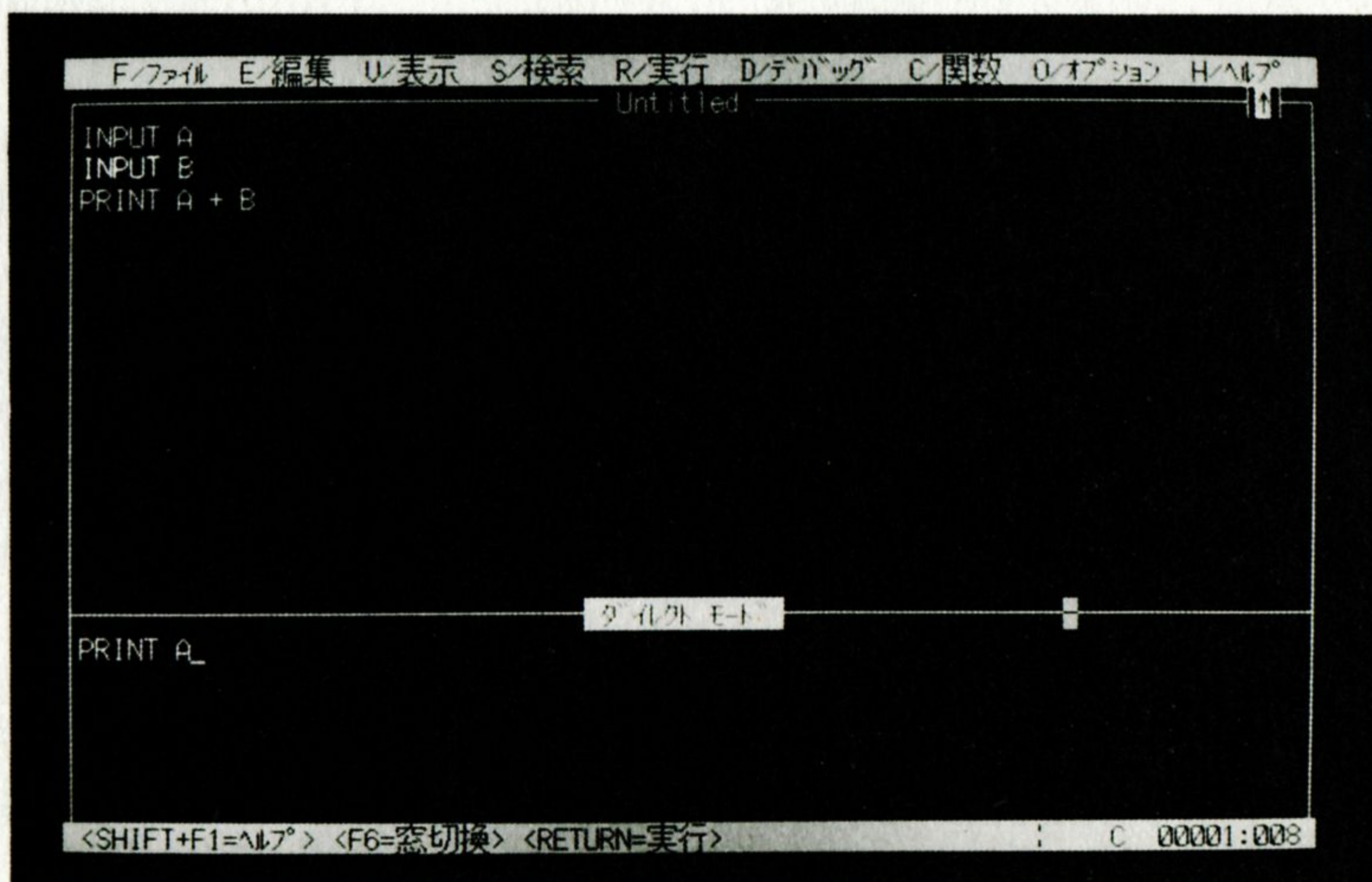
ダイレクトモード

プログラムの動きなどをちょっと試してみたい、などというときのために、Quick BASICでは「ダイレクトモード」が用意されています。

エディタ画面は、画面の上部の大部分が「ビューウィンドウ」と呼ばれ、エディタ画面のいちばん下の部分が、ダイレクトモードのウィンドウで構成されています。

ダイレクトモードへ移るには、ファンクションキーの[f・6]に割り当てられている「窓切替」を使います。

[f・6]を押すと、ビューウィンドウの外側にあった白いゲージが消え、「ダイレクトモード」というタイトルバーが白く反転して、ダイレクトモードに移っ

```
F/ファイル E/編集 U/表示 S/検索 R/実行 D/デバッグ C/関数 O/オプション H/ヘルプ
Untitled
INPUT A
INPUT B
PRINT A + B

ダイレクトモード
PRINT A_

<SHIFT+F1=ヘルプ> <F6=窓切換> <RETURN=実行>     C 00001:008
```

■ダイレクトモードの画面

たことがわかります。

カーソルもダイレクトモードのウィンドウに移動していますので、この位置から入力が行われます。

それでは試しに、

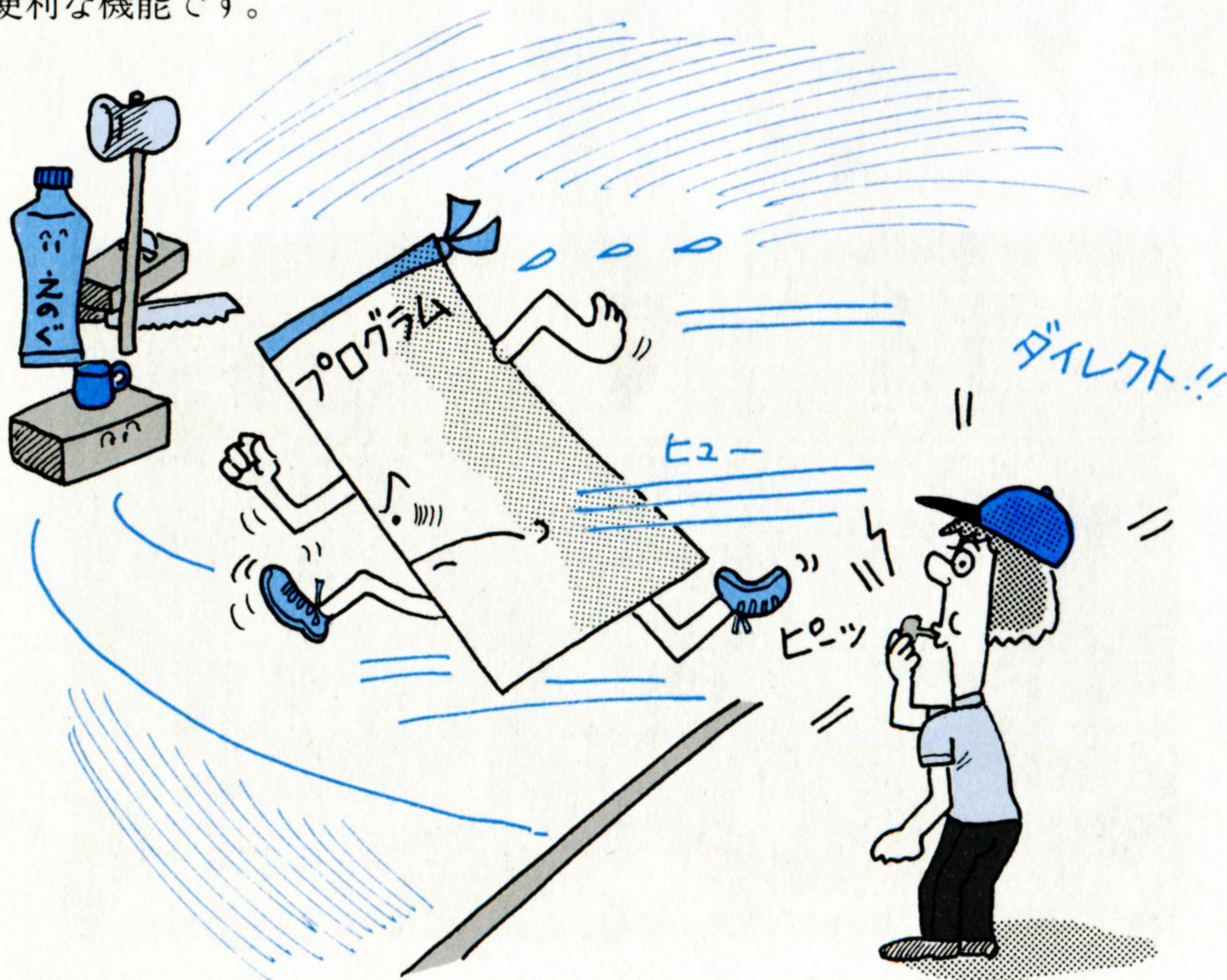
PRINT　A ⏎

と入力してみましょう。

画面が実行画面に切り替わり、変数Aに代入されている値の「3」が表示されています。画面のいちばん下には、「何かキーを押してください。」と表示されています。

スペースを押すと、またダイレクトモードの画面に戻ります。これで、変数Aの内容がわかりました。

ダイレクトモードは、変数の内容を調べたり、変更したりしてプログラムのテストランに利用します。

また、プログラムのエディタ中の画面を乱さずに実行画面の表示位置などを調整したり、ステートメントを試してみたりするのにも利用することができる便利な機能です。

プログラムの保存と呼び出し

同じプログラムを何度も入力するのはたいへん

プログラムの保存

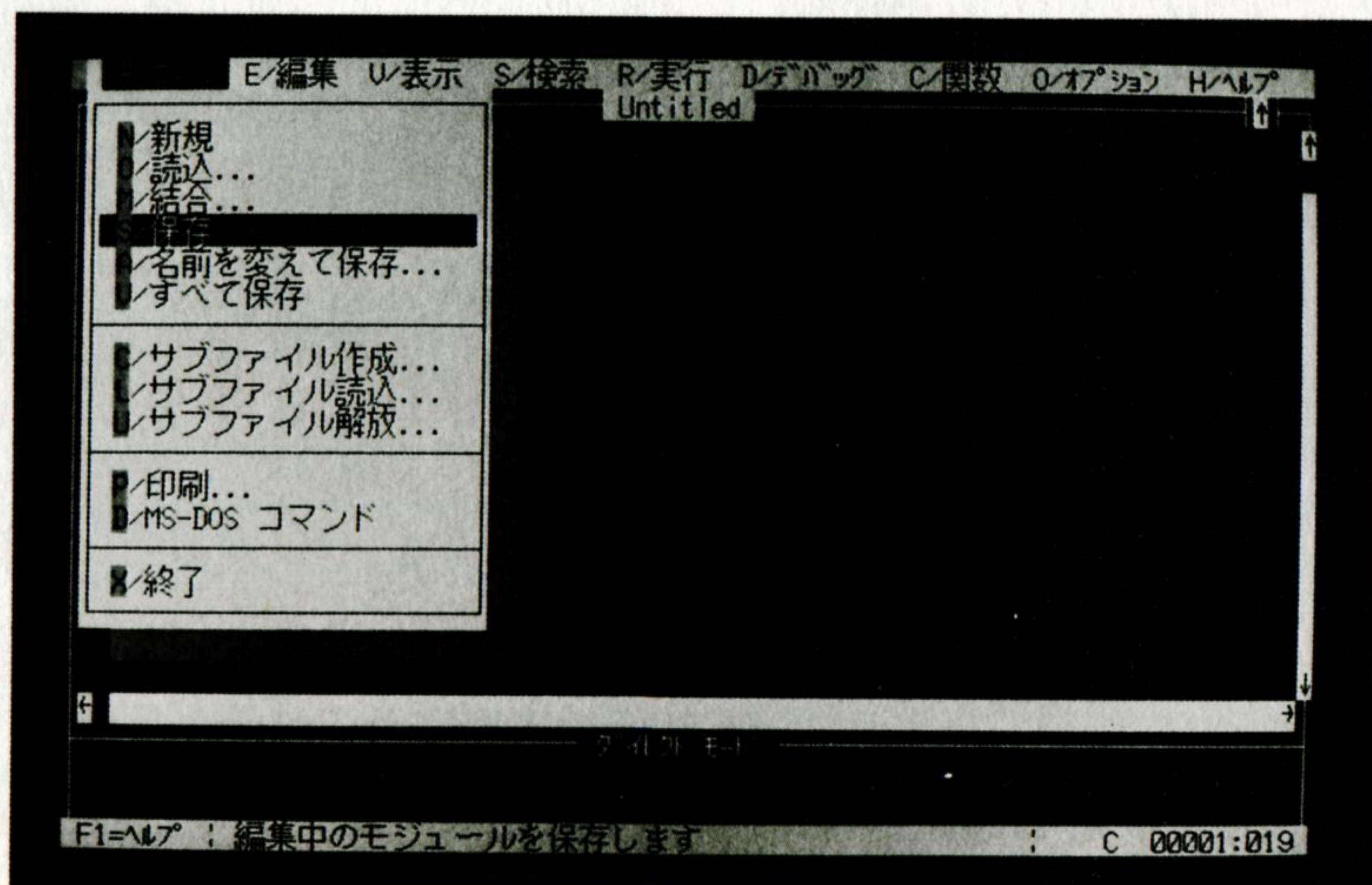
せっかくつくったプログラムも、フロッピーディスクに保存しておかないと、Quick BASICを終了したり、パソコンの電源を切った瞬間にすべてが消えてしまいます。

できたプログラムをフロッピーディスクに保存しておかないと、同じプログラムを何度も何度もキーボードからタイプしなければなりません。

短くて簡単なプログラムならたいしたことはありませんが、このあとに出てくるサンプルプログラムのように、かなり長いものになると、プログラムの入力途中で時間がなくなって一時中断しなければならなくなるかもしれません。

こんなとき、プログラムを保存しておかないと、初めからやり直しになってしまうのでたいへんです。

前にあった例題もフロッピーディスクに保存してみましょう。

メニューバーから「F／ファイル」を選択します。次に、「S／保存」を選択

■メニュー画面　F／ファイル、S／保存

します。

これからは、このようなメニューとコマンドの選択をひとまとめにして、

F／ファイル

S／保存

のように表わします。

保存のコマンドを選択すると、画面に「ダイアログボックス」が表示されます。ダイアログボックスは、Quick BASICが必要な情報を求めてくる対話の窓口です。

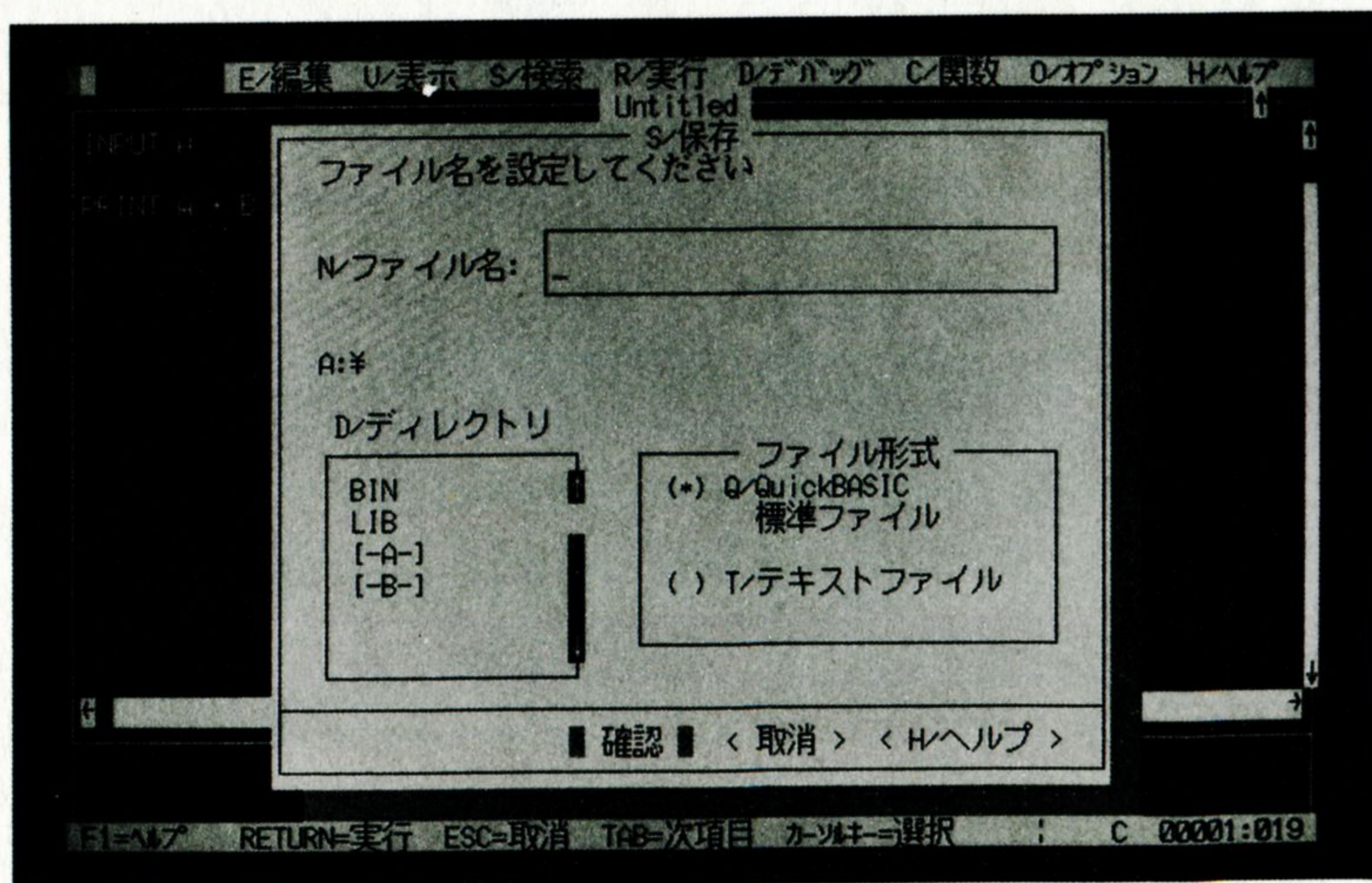
ここでは、フロッピーディスクに保存するときのファイル名をたずねてきます。

「**N／ファイル名**」の位置のボックスのなかにカーソルがあることを確認してください。もしなければ、[TAB]キーを何回か押してカーソルを移動するか、マウスカーソルをボックスのなかに移動して左クリックします。

キーボードからファイル名を入力します。

REIDAI ⏎

とすると、次に「確認」のボックスにカーソルが移ります。まちがえていたら、[TAB]キーとカーソルキーを使ってカーソルを移動して修正します。

まちがいがなくなれば、「確認」のボックスで⏎、または、マウスを左クリックします。

■ダイアログボックスの表示

S／保存　F／ファイル
ダイアログボックス　X／終了
N／ファイル名　QB

フロッピーディスクがカチャカチャと音を出して、プログラムが記録されているのがわかります。

画面はまた、エディタ画面に戻ります。

終了と再起動

これでQuick BASICを終了してしまってもだいじょうぶです。

Quick BASICの終了は、

F／ファイル

X／終了

のメニューを選択します。

Quick BASICは、終了する前にまだ保存していないプログラムがある場合には、ダイアログボックスを開いて、プログラムを保存して終了するかどうかたずねてきます。

今回はすでに保存がすんでいるので、画面をクリアして、MS-DOSのコマンド入力待ちの状態に戻ってしまいます。

プログラムの呼び出し

ディスプレイ画面はすでに、Quick BASICを終了して、MS-DOSの管理下になっているので、

A>

のプロンプトが表示されて、MS-DOSのコマンド待ちの状態になっています。

ここで、プロンプトにつづいて、

QB⏎

と入力してください。

また、フロッピーディスクがカチャカチャといって、Quick BASICが再起動されます。Quick BASICの初期画面のエディタ画面が表示されたら、先ほどつくった「REIDAI」のプログラムを呼び出してみましょう。

メニューから

F／ファイル

O／読込

を選択します。

すると、ファイル名入力のダイアログボックスが開かれます。

TAB キー、またはマウスを使ってファイル名の表示されている大きなボックスに反転表示されているカーソルを移動して、読み出すプログラムを選択します。

REIDAI. BAS

に反転表示のカーソルを合わせます。

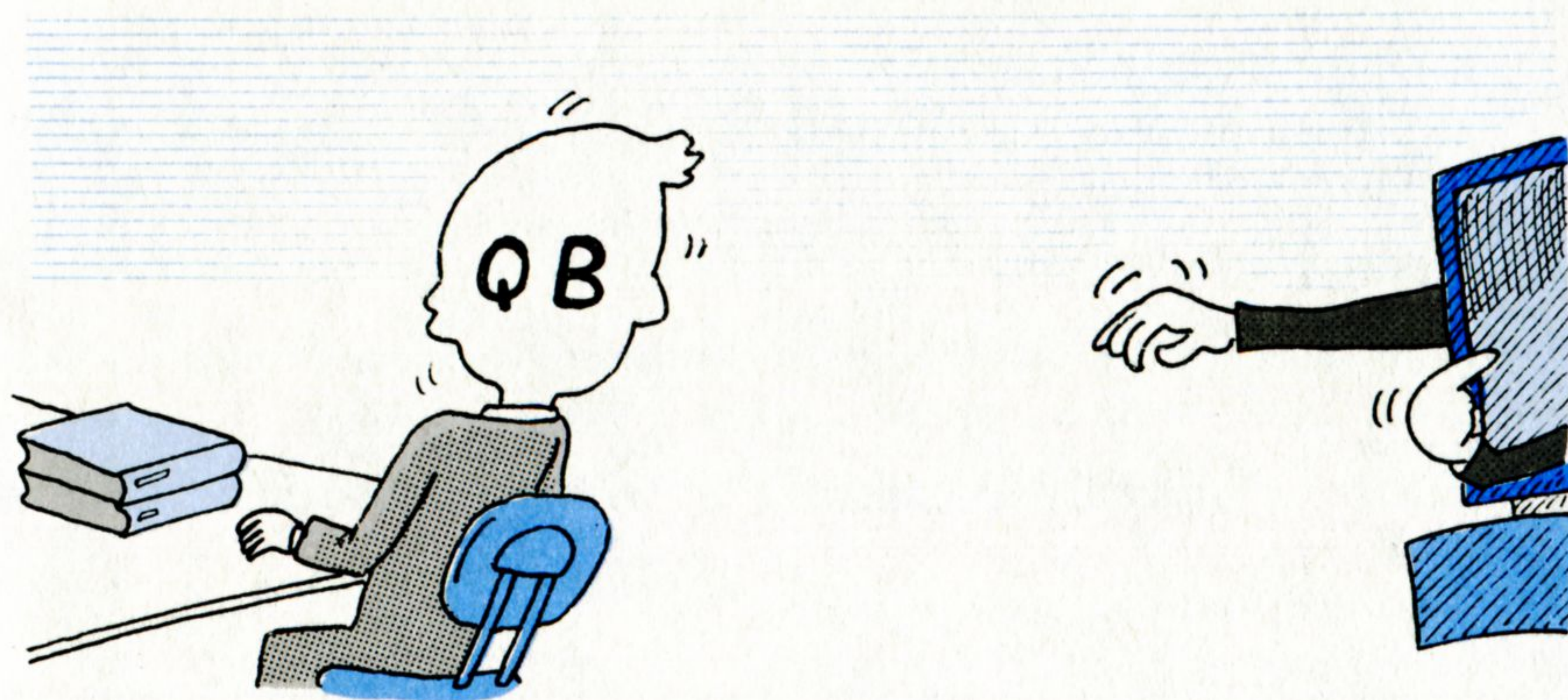
先ほどは、「REIDAI」というファイル名だけで保存しましたが、Quick BASICが自動的に、BASICのプログラムファイルであることを示す「.BAS」の拡張子を付加しているのがわかります。

ここで、⏎またはマウスの左クリックをすると、フロッピーディスクからプログラムを読み出して、エディタ画面になります。

このあとは、またプログラムに手を加えたり、実行したりすることができるようになります。

このようにQuick BASICでは、メニュー選択でプログラムの作成、テストラン、修正、保存、呼出、終了などが簡単にできます。

また、エディタ画面と実行画面がきちんと分かれているので、混乱なくプログラムの作成と、デバッグを行うことができます。

Quick BASICをひと通りごく簡単になぞってみましたが、いかがですか？きっとQuick BASICの環境が気に入ったことでしょう。

PART○2

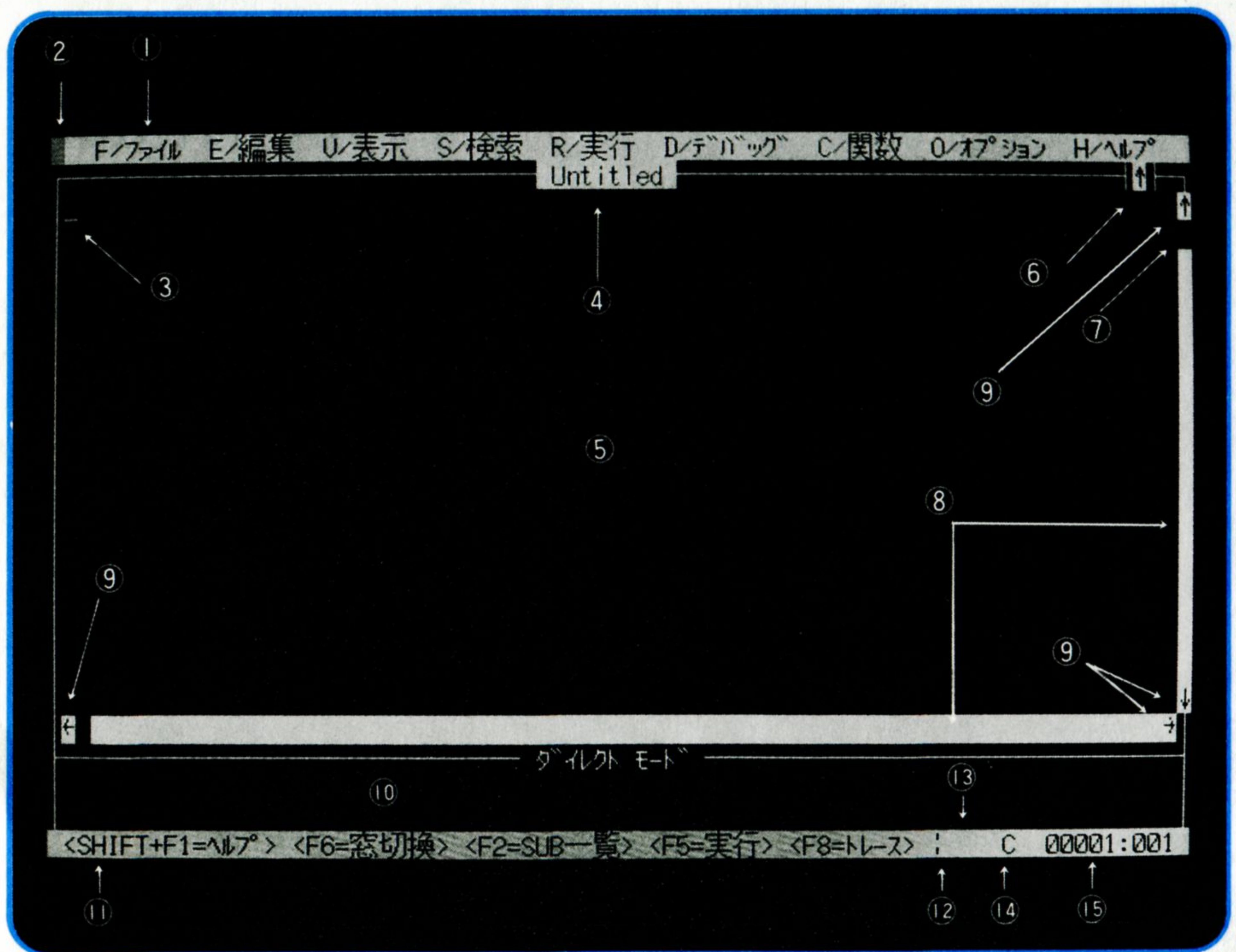
画面と応用いろいろ

Quick BASIC のウィンドウ

Quick BASIC の初期画面

Quick BASIC をスタートすると、下のような画面が表示されます。

①メニューバー
②マウスカーソル
③カーソル
④タイトルバー
⑤ビューウィンドウ
⑥フル画面表示ボックス
⑦スクロールボックス
⑧スクロールバー
⑨スクロールアイコン

F/ﾌｧｲﾙ E/編集 V/表示 S/検索 R/実行 D/ﾃﾞﾊﾞｯｸﾞ C/関数 O/ｵﾌﾟｼｮﾝ H/ﾍﾙﾌﾟ
Untitled
ﾀﾞｲﾚｸﾄ ﾓｰﾄﾞ
<SHIFT+F1=ﾍﾙﾌﾟ> <F6=窓切換> <F2=SUB一覧> <F5=実行> <F8=ﾄﾚｰｽ> C 00001:001

⑩ダイレクトウィンドウ
⑪参照バー
⑫コマンドインジケータ表示位置
⑬カナキーロック表示位置
⑭キャプスキーロック表示位置
⑮行／列カウンタ

表示各部の名まえと働き

①メニューバー　コマンドメニューの名まえが表示されています。

②マウスカーソル　マウス使用時のマウスの示している画面上の位置。机の上でマウスを動かせばそれに応じて移動します。

③カーソル　キーボードから入力される文字の記入される位置。[↑][↓][←][→]のカーソルキーで移動させます。

④タイトルバー　ビューウィンドウに表示されているプログラムのファイル名。

⑤ビューウィンドウ　編集対象になっているプログラムのリストが表示されるウィンドウ。

⑥フル画面表示ボックス　マウスで左クリックすると、ビューウィンドウのサイズを最大に広げます。キーボードからは[CTRL]+[f･10]。

⑦スクロールボックス　プログラム中でのカーソルの位置を相対的に示します。

⑧スクロールバー　マウスで左クリックすると、プログラムテキストがマウスカーソルの指す相対的位置まで順次スクロールします。

⑨スクロールアイコン　マウスで左クリックすると、プログラムテキストが一度に1行ずつ上下に、または1文字ずつ左右にスクロールします。

⑩ダイレクトウィンドウ　BASICのステートメントを直接実行するためのウィンドウ。

⑪参照バー　よく使われるショートカットキーを表示しています。マウスで実行したいショートカットキーを選択して、左クリックすると、実行することができます。[GRPH]キーを押してメニュー選択すると、メニューの解説が表示されます。

⑫コマンドインジケータ表示位置　カーソルを指定した位置にジャンプさせるための、プレスマーカを設定している場合には「^K」を、ワードスターライクなキー操作を行っている場合は「^Q」を表示します。

⑬カナキーロック表示位置　[カナ]キーが押されて、カナキーロックの状態のときに「カ」が表示されます。

⑭キャプスキーロック表示位置　[CAPS]キーが押されて、キャプスキーロックの状態のときに「C」が表示されます。

⑮行／列カウンタ　編集対象のプログラムのリスト中でのカーソルの位置を「行：列」で示します。

4つのウィンドウがまるでコクピットのように

Quick BASICでプログラム作成中は、「ビューウィンドウ」、「ダイレクトモード」の2つのウィンドウが、ディスプレイ画面の上部と下部に開かれています。

プログラムを実行するときには、隠れていた実行画面が現れ、また、プログラムのデバッグ中には、ウォッチウィンドウが現れて、ぜんぶで4つの画面があることになります。

まるで、航空機のコクピットさながらに、Quick BASICの世界をあなたが自由自在に操れるようになっているのです。

ビューウィンドウ

2つの部分を同時に編集できるプログラム作成

ビューウィンドウは、Quick BASICを起動したときに現れるエディタの画面で、このウィンドウを使ってプログラムを作成します。

プログラムを編集するためのコマンドや、デバッグするためのコマンドがいちばん上のメニューバーに配置されています。

プログラムが読み込まれると、初めに「モジュールレベルコード」と呼ばれ

ジェット機のコクピットで操縦するように4つの画面を自在に操作

るプログラムのメイン（主）部分が表示されます。

ビューウィンドウは、上下２つに「分割」して、プログラムの２つの部分を同時に編集対象とすることができます。

カーソルが置かれていて、現在操作対象となっているウィンドウを「アクティブウィンドウ」といいます。

アクティブウィンドウは、大きさを変えることができます。

●ビューウィンドウの操作

f・2	SUB・FUNCTION プロシージャの呼び出し
SHIFT ＋ f・2	次の SUB・FUNCTION プロシージャの呼び出し
↑ ↓ ← →	ウィンドウの上下左右の端で押すとスクロール
GRPH →V	ビューウィンドウの上下２分割
f・6	アクティブウィンドウの変更
SHIFT ＋ f・6	アクティブウィンドウの変更
GRPH ＋ ＋	アクティブウィンドウの拡大
GRPH ＋ －	アクティブウィンドウの縮小
CTRL ＋ f・1	アクティブウィンドウの最大面積への拡大、戻し

ダイレクトモード

ちょっと試しにすぐ実行

ディスプレイ画面の下部に位置するウィンドウが、ダイレクトモードのウィンドウです。

ダイレクトモードでは、BASICのステートメントをキーボードから入力したり、編集コマンドを使ってビューウィンドウから複写して、それをすぐ実行してみることができます。

LOCATEステートメントでの位置の調整を、実行画面で実際に試してみたり、プログラムのデバッグ中に変数に値を代入してみて、動作を確かめたりすることができます。

ダイレクトモードでは、１行255文字、10行までのステートメントを「：」（コ

ロン）で区切って書くことができます。

ステートメントの実行は、実行するステートメントの行にカーソルを合わせて、↵キーを押して行います。実行結果は実行画面に表示されます。

●ダイレクトモードの操作

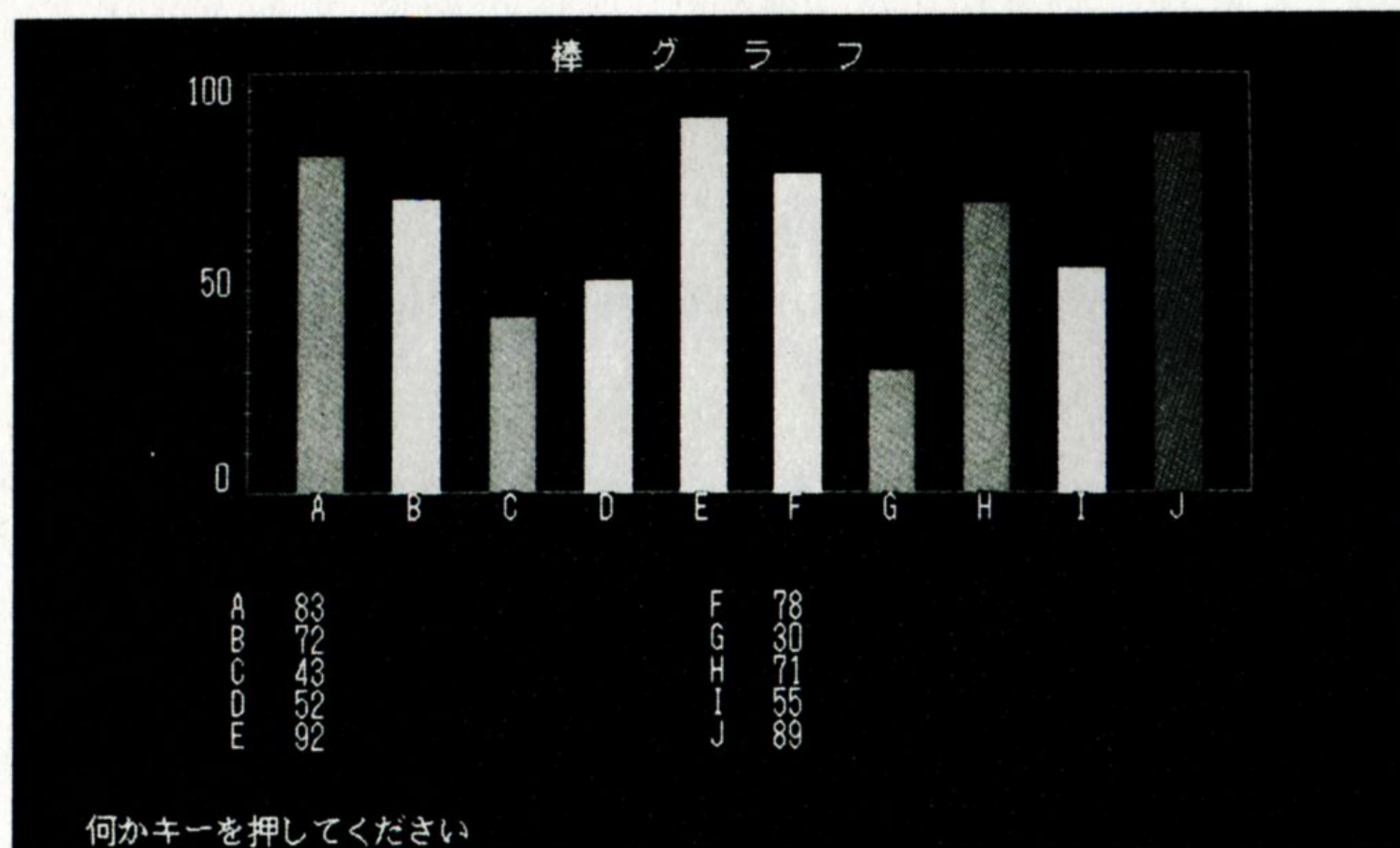
f・6	ダイレクトモードをアクティブにする
SHIFT ＋ f・6	前のウィンドウに戻る
↵	ステートメントの実行

実行画面

実行画面は、Quick BASICのプログラムを実行したり、ダイレクトモードでステートメントを実行したときに、画面が切り替わって現れます。

Quick BASICのプログラムを編集している画面とはまったく別の画面で、プログラムを実際に動作させたときのMS-DOSが管理している画面と同じになります。プログラムやステートメントの実行が終わると、「何かキーを押してください」と画面の最下部に表示されますので、スペースバーなどのキーを押すと、ビューウィンドウに戻ります。

●実行画面の操作

f・5	プログラムの続行
SHIFT ＋ f・5	プログラムの実行
↵	ダイレクトモードでのステートメントの実行
スペース	ビューウィンドウへの復帰

■実行画面

Quick BASIC との対話

メニューを開いてコマンドを選んで実行！

メニューとコマンド

Quick BASIC では、プログラムの作成や保守に必要なコマンドを、ディスプレイ画面のメニューバーのなかから選択して実行できるようになっています。

コマンドは、項目ごとに分類されて、画面最上段のメニューバーに配置されています。

コマンドを実行するには、分類されたメニューを開いて、コマンド一覧のなかから選択して実行することができるようになっています。

メニューバーには次のような項目が配置されています。

●メニュー

F/ファイル　プログラムファイルの読み出しや保存、リストの印刷、Quick BASIC の終了などのコマンドが配置されています。

E/編集　プログラムを作成するのに必要な文字のコピー、削除やカット、ペースト、構文チェックの指定などのコマンドが入っています。

V/表示　ビューウィンドウの分割やプロシージャの表示、実行画面の表示などの画面表示のコマンドが配置されています。

S/検索　プログラムのなかから特定の文字列を探し出す検索や、文字列の置き換え、指定したラベルを検索するコマンドが入っています。

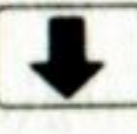
R/実行　プログラムの実行、再実行、続行やプログラムをコンパイルして保存するコマンドなどが配置されています。

D/デバッグ　プログラムのデバッグに必要なウォッチ・ウォッチポイント・ブレークポイントの設定や解除、トレースやヒストリの設定のコマンドが入っています。

C/関数　このメニューはコマンドが配置されているのではなく、実行中のプロシージャが記録されていて、プロシージャ間の呼

び出しの状況が把握できます。また、選択したプロシージャまでを実行することもできます。

O/オプション　画面表示やサーチパスの設定、マウスの右ボタンの機能、構文チェック、FullメニューとEasyメニューの切り換えなどのオプション機能を設定します。

H/ヘルプ(F･1)　ヘルプ（操作説明）を画面に呼び出します。

マウスとキーボード

Quick BASICでは、キーボードだけではなく、マウスを使ってコマンドを実行することができます。

メニューバーのなかから、必要なコマンドの入っているメニュー項目を選択して、プルダウンメニューを開きます。

プルダウンメニューのなかから、コマンドを指定して実行します。

キーボード、マウスともにこの手順でコマンドを指定します。

キーボードは GRPH キーと↑↓←→のカーソルキーを使って操作しますが、画面の切り替えにファンクションキーを使ったりしなければならないために、少し使いにくいかもしれません。

マウスなら直接コマンドやメニューを選択できます。プログラムの作成、編集中には、マウスを使って画面を自由に行き来して操作するのが快適でしょう。

プロシージャ　X／終了
プルダウンメニュー　メニューバー
マウスカーソル　F／ファイル

●マウスで操作

マウスで操作する場合は、マウスをてのひらに軽く包んで、机のような平らなところで動かします。手の動きに合わせて、マウスカーソルがディスプレイ画面上を移動します。

メニュー選択　メニューバーなどの選択したいメニューやコマンドにマウスカーソルを合わせて左クリック（左ボタンを1回押す）します。

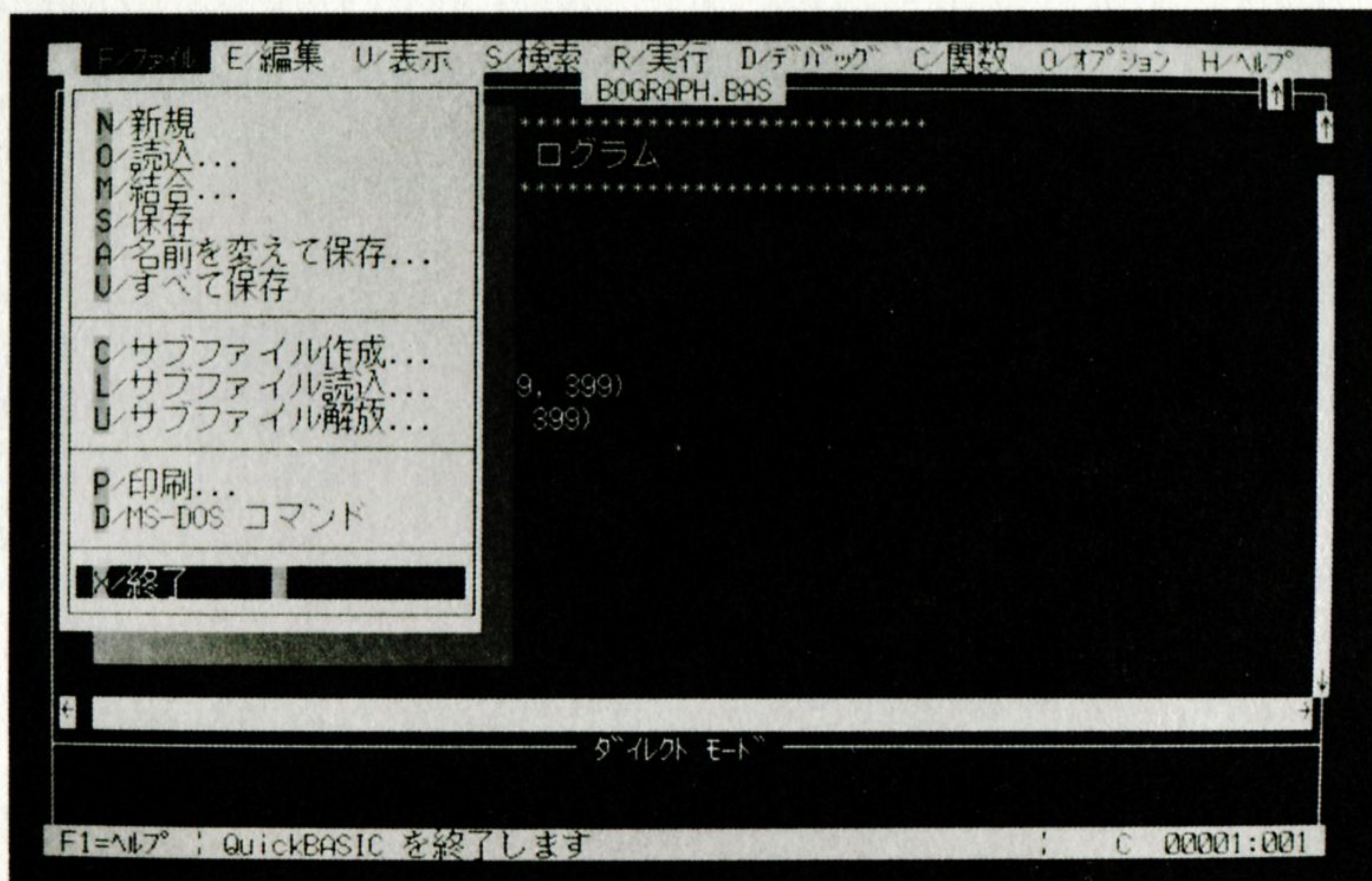
カーソルの移動　ビューウィンドウのなかなどでは、プログラムリストの上を動き回って、マウスカーソルがカーソルを移動させたいところにきたら、左クリックします。すると、カーソルがその位置に移動してきます。

●マウス操作の実際

マウスで実際に、「F／ファイル」のメニューから「X／終了」のコマンド選択の操作をしてみましょう。

メニューバーからの選択

マウスを机の上で転がして(マウスを移動することを転がすといいます)、紫色のマウスカーソルを、画面最上部のメニューバーに合わせます。

メニューバーのなかからF／ファイルの項目にマウスカーソルを移動します。

選択画面

ここで、**左クリック**すると、プルダウンメニューが表示されて、「F／ファイル」に分類されたコマンドが表示されます。

コマンドを選択するためにマウスを動かして、マウスカーソルをいま開かれたプルダウンメニューのなかに移動します。

ここでは、いちばん下のX／終了にマウスカーソルを合わせます。

マウスを**左クリック**すると、コマンドが選択され、終了コマンドが実行されます。Quick BASICが終了して、MS-DOSに戻ります。

マウスは、手の動きとディスプレイ画面上のマウスカーソルの動きが連動しますし、画面上のアイコンと呼ばれるマークやメニューが次つぎに変化していくので、直観的に操作できて便利です。

●キーボードで操作

キーボードからもマウスと同じように、実際に「F／ファイル」のメニューから「X／終了」のコマンド選択の操作をしてみましょう。

メニューバーからの選択

まずコマンド選択の状態にはいるために、GRPHキーを押します。

画面最上部のメニューバー中のメニューのうちのひとつが黒地に黄色文字の反転になってアクティブになるので、メニュー選択が可能な状態になったことがわかります。

メニューバーのなかから「F／ファイル」の項目がアクティブになるように、カーソルキーの➡⬅を使って、アクティブ項目を合わせます。

「F／ファイル」がアクティブになったら、⏎キーを押します。プルダウンメニューが表示されて、分類されたコマンドが表示されます。

コマンドを選択するために、プルダウンメニューのなかのコマンドのひとつがアクティブになって、コマンド選択が可能な状態になっていることがわかります。ここでカーソルキー⬆⬇を押します。

押すたびに選択可能なコマンドが次つぎとアクティブになり、反転表示されるのがわかります。ここでは、いちばん下の「X／終了」コマンドをアクティブにします。

⏎すると、コマンドが選択され、終了コマンドが実行されます。

Quick BASICが終了して、MS-DOSに戻ります。

キーボードからの操作は、ほとんどカーソルキーと⏎キーでできるようになっていますが、より簡単にアルファベットの文字キーを使って操作することが

アイコン	X／終了
アクティブ	ホームポジション
F／ファイル	GRPH・F・X

できるようにもなっています。

覚えてしまえば、カーソルキーを使わないため、手をキーボードのホームポジション※から動かさなくてすみますから、コマンドの実行がより早くできるようになります。

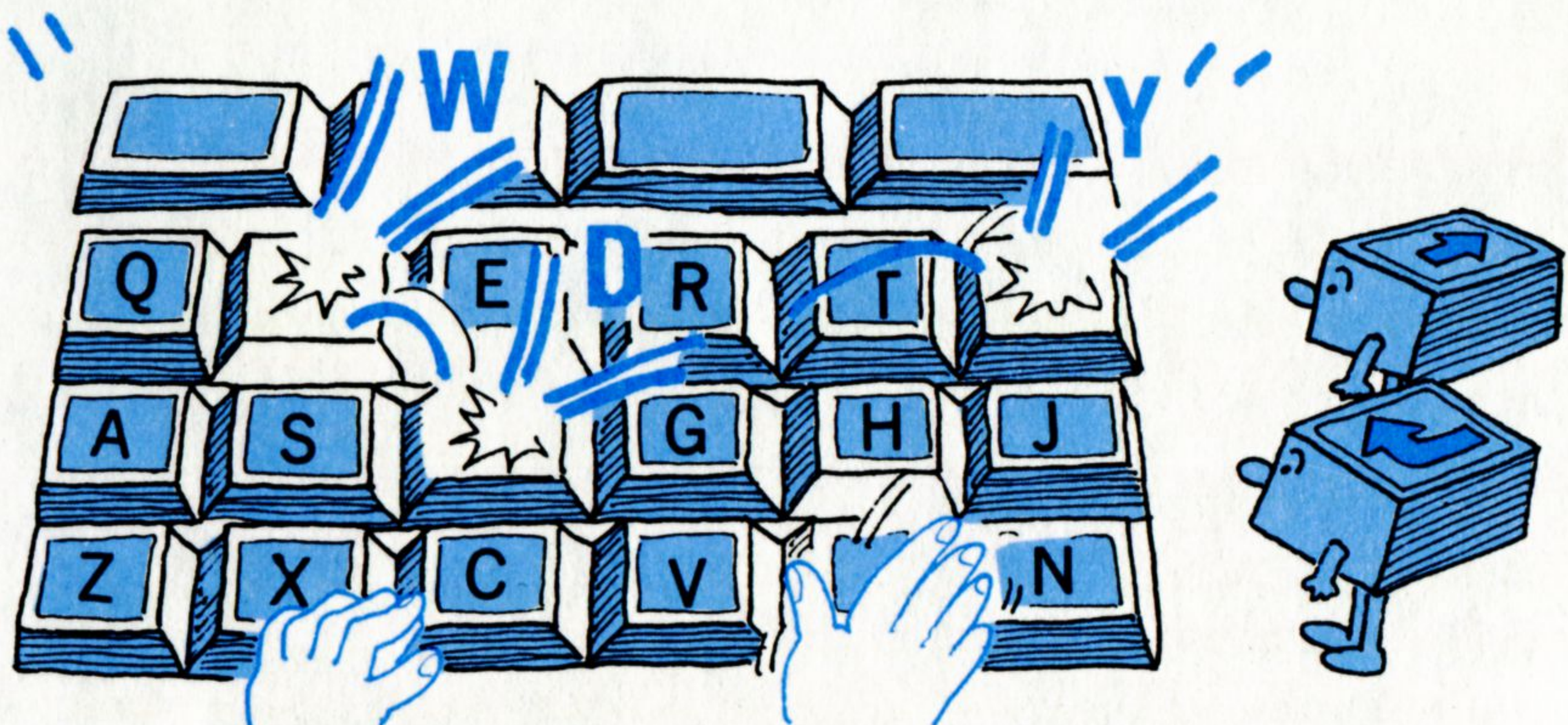
●文字入力でのコマンド選択

実際に「X／終了」のコマンド選択を、キーボードからの文字入力でやってみましょう。

GRPH、F、Xの3つのキーボード操作だけで終わってしまいます。

このような操作を簡単に、GRPH・F／ファイル・X／終了などと表現することにしましょう。

ショートカットキーの簡単操作

Quick BASICのコマンド実行の方法には、ファンクションキー一発で実行できる「ショートカットキー」操作が用意されています。

※ホームポジション　キーボードからタイプするときは、一般的に左右10本の指を使います。

このとき、各キーを押す指が決まっていて、なんのキーも押さない準備のときの指の位置をホーム位置、あるいはホームポジションといいます。

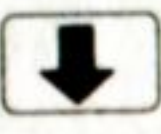
キーボードを見ないでもタイプできるブラインドタッチという方法を身につければ、キーボードからの入力も素早く行うことができます。

これは、比較的よく使われるコマンドをファンクションキーに割り当てて、すぐに実行できるようにしてあるものです。

f・1	キーワードヘルプ	SHIFT＋f・1	ヘルプの利用法
f・2	SUB 表示	SHIFT＋f・2	次の SUB
f・3	次を検索		
f・4	実行画面		
f・5	続行	SHIFT＋f・5	実行
f・6	窓切り替え（後）	SHIFT＋f・6	窓切り替え（前）
f・7	現在行まで実行		
f・8	1行実行（関数）	SHIFT＋f・8	ヒストリ（後）
f・9	ブレークポイント	SHIFT＋f・9	簡易ウォッチ
f・10	1行実行	SHIFT＋f・10	ヒストリ（前）

Quick BASIC には、ファンクションキーの上に置いて使う「テンプレート」が付属しています。これを使えばひと目でショートカットキーの割り当てがわかります。

ダイアログボックス

コマンドを実行する場合、詳しく実行内容を指定しなければならないものがあります。

例えば、プログラムをキーボードから打ち込み終わって、フロッピーディスクに保存するときには、少なくともファイル名を指定しなければなりません。

このようなときに、Quick BASIC は「ダイアログボックス」を開いて、必要な項目の指定を求めてきます。

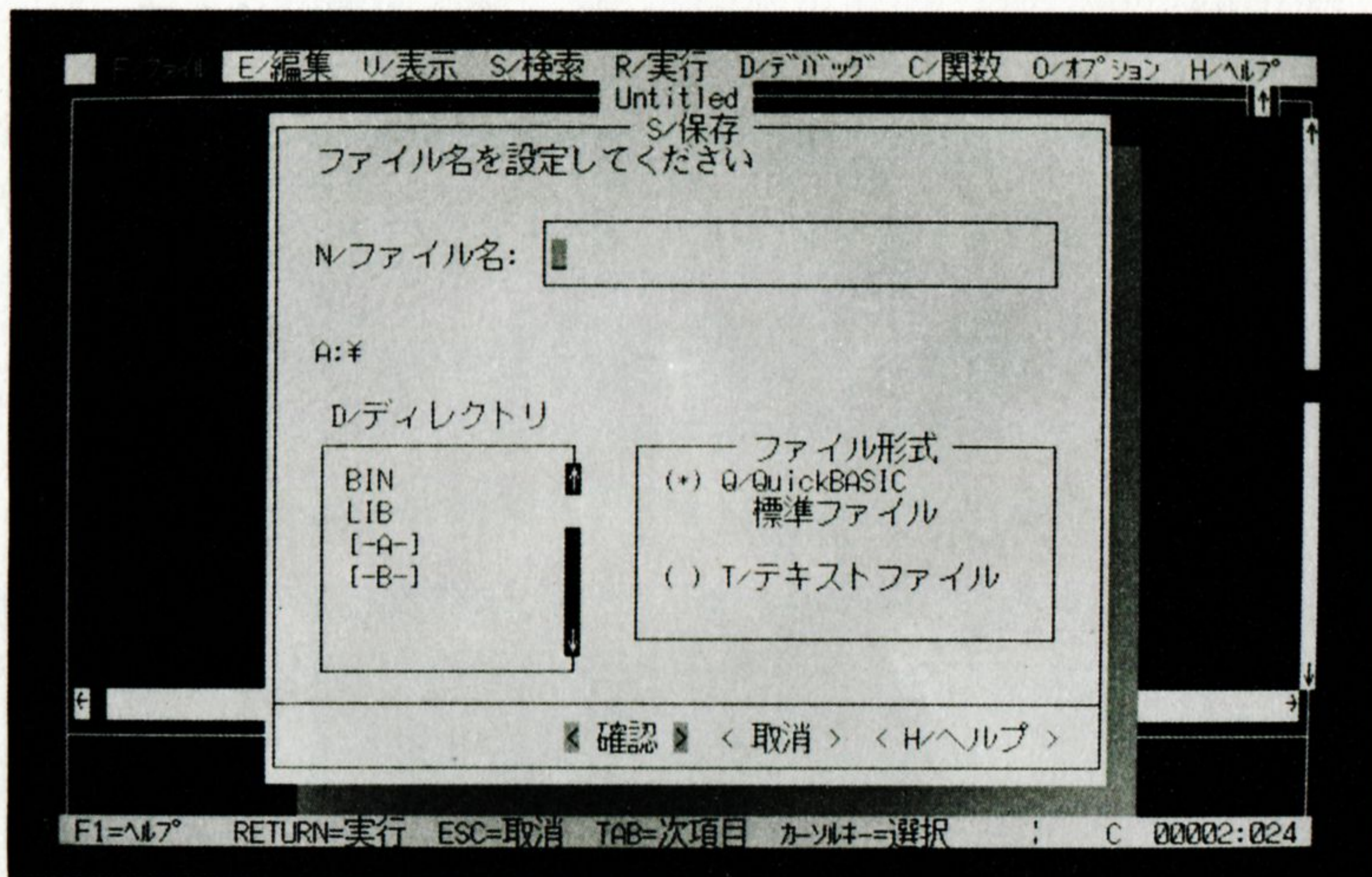
また、Quick BASIC を終了する場合でも、ビューウィンドウにプログラムの変更があったテキストがある場合には、ダイアログボックスが開いて、Quick BASIC からプログラムの保存をたずねられます。

このときに保存を指定すると、重ねて「S／保存ダイアログボックス」が現れて、Quick BASIC から対話を求められます。

「N／ファイル名」には、保存するプログラムのファイル名をキーボードから入力します。

保存形式には、「Q／標準ファイル※」と「T／テキストファイル※」があります。マウスカーソルを合わせて左クリック、またはカーソルキー⬆⬇で、

	S／保存ダイアログボックス	T／テキストファイル
	N／ファイル名	確認
ショートカットキー	Q／標準ファイル	取消

```
F/ファイル E/編集 V/表示 S/検索 R/実行 D/デバッグ C/関数 O/オプション H/ヘルプ
Untitled
S/保存
ファイル名を設定してください

N/ファイル名: [              ]

A:¥

D/ディレクトリ          ファイル形式
BIN                    (*) Q/QuickBASIC
LIB                        標準ファイル
[-A-]
[-B-]                  ( ) T/テキストファイル

< 確認 > < 取消 > < H/ヘルプ >
F1=ヘルプ RETURN=実行 ESC=取消 TAB=次項目 カーソルキー=選択 : C 00002:024
```

■ダイアログボックス（保存）

選択可能を示すアクティブポインタ「＊」(アスタリスク) を移動、⏎キーを押して、どちらかを選択します。

最後に指定内容の確認があります。よければ「確認」にマウスカーソルを合わせて左クリック、またはカーソルキー⬅➡で、アクティブポインタを移動して、どちらかを選択します。

訂正があるときは、「取消」を選択すればもとの状態に戻ることができます。

ウィンドウのスクロール

ウィンドウは、一度に表示できる行、列が制限されています。大きなプログラムの場合は、当然リストのぜんぶを表示することはできません。小さな窓を開けてプログラムの一部分を表示させているに過ぎません。

※標準ファイル　Quick BASIC 独自の内部コードの形式でフロッピーディスクなどに保存されるファイル。MS-DOS の TYPE コマンドやエディタなどでは読むことができません。
※テキストファイル　一般的な MS-DOS のテキストファイルの形式（ASCII 形式）で、フロッピーディスクなどに保存されるファイル。MS-DOS の TYPE コマンドやエディタなどでも読むことができます。

この窓に、プログラムのリストを次つぎと表示させるためにスクロールの機能が用意されています。

●マウスでのスクロール

マウスで画面をスクロールするために、3つの方法が用意されています。

スクロールアイコン	マウスカーソルをスクロールアイコンに置いて、左クリックすると矢印の方向に画面がスクロールする。
スクロールボックス	マウスカーソルをスクロールボックスのなかに合わせ、左ドラッグでスクロールボックスを移動すると、それに合わせてビューウィンドウのリストがスクロールする。
スクロールバー	スクロールバーのなかのスクロールボックスが、表示中のリストのテキスト全体での相対的位置を示している。相対的にスクロールする場合には、スクロールボックスの上下左右のスクロールバーを左クリックして、リストをスクロールすることができる。

●キーボードでのスクロール

[↑][↓][←][→]のカーソルキーをウィンドウの上下左右の端で押しつづけると、カーソルの逆方向にスクロールします。

1画面ずつスクロールしたいときは、

[ROLL UP]……1画面下にスクロール

[ROLL DOWN]……1画面上にスクロール

のキーで行うことができます。

スクロールアイコン
スクロールボックス
スクロールバー

ウィンドウの切り替え

ビューウィンドウを2つに分割すると、プログラムの2つの部分を並行して編集することができます。

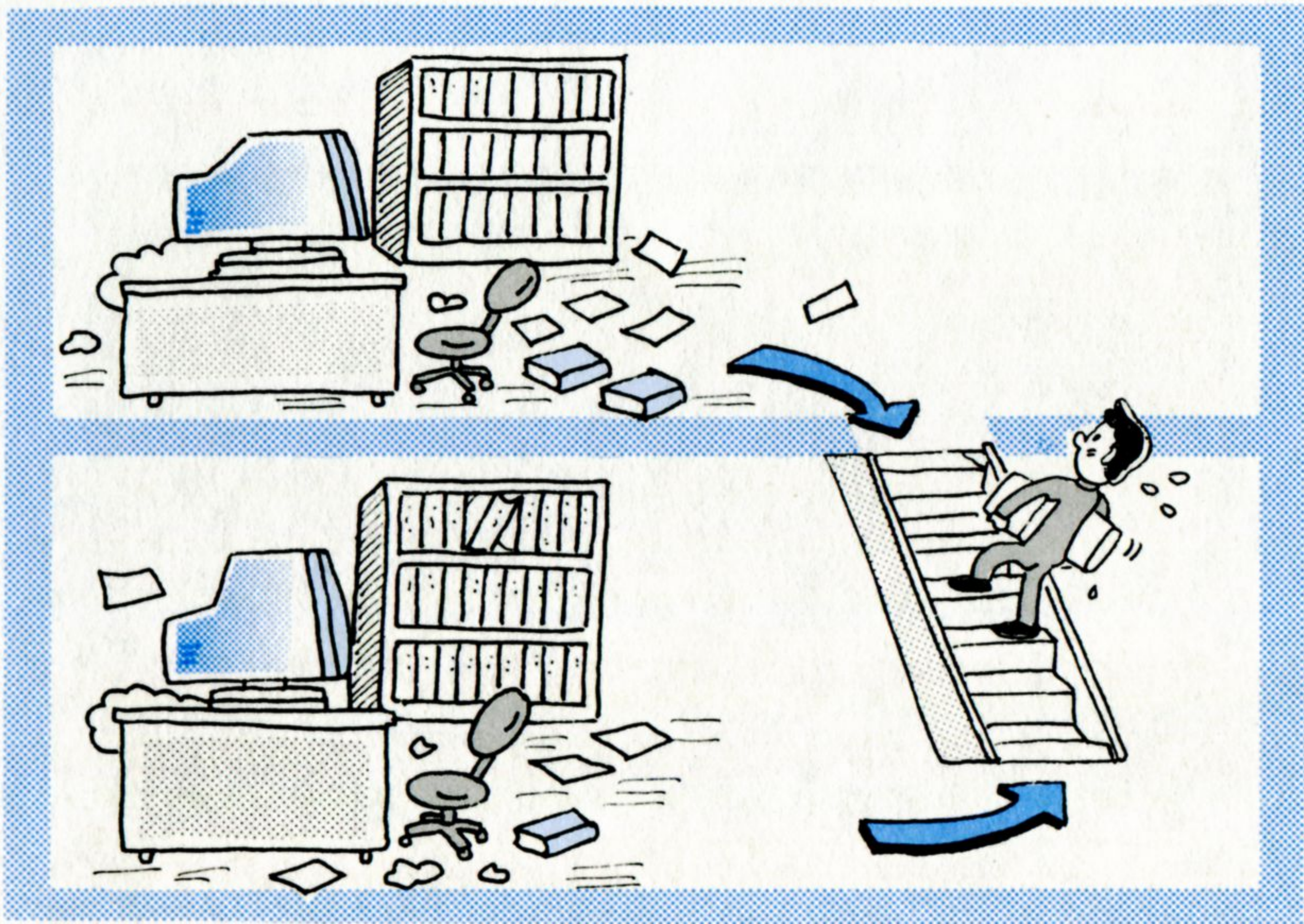
画面は上下に分かれますから、この2つのウィンドウを行ったり来たりしながら、プログラムを編集していくことになります。ダイレクトモードのウィンドウにも同様に移動することができます。

3つのウィンドウを切り替える方法は同じです。

●マウスでのウィンドウの切り替え

マウスカーソルを移動したいウィンドウ内にもっていって、左クリックするだけで、ウィンドウが切り替わり、マウスカーソルの位置にカーソルが移動します。

●キーボードでのウィンドウの切り替え

f・6または SHIFT ＋f・6で、相互にウィンドウが切り替わります。

お助けマンは「ヘルプ」

困ったときはなんでも「ヘルプ」が教えてくれる

オンラインヘルプ／QBアドバイザ

Quick BASICでは、操作法やBASICのステートメントがわからなくなった場合に、マニュアルを引かなくてもパソコンの画面の上で調べられる「オンラインヘルプ」と「QBアドバイザ」が用意されています。

オンラインヘルプ、QBアドバイザは、ヘルプ機能として統一されて、Quick BASICの操作法やステートメント、関数などの解説をディスプレイ画面上に表示します。

また、Quick BASICのマニュアル機能を、プログラミングの最中に画面上に呼び出して、関連する項目を参照したり、プログラムの具体例を見たり、サンプルプログラムを作成中のプログラムに複写して利用することができます。

このような機能を「ハイパーリンク機能」といいますが、そのおかげで本になったマニュアルよりずっと使いやすくなっています。

ヘルプ機能は、メニューバーの「H／ヘルプ」の項目に設定されています。

キーボードからは、GRPHキーを押して、メニュー選択の状態になってか

```
F/ファイル  E/編集  V/表示  S/検索  R/実行  D/デバッグ  C/関数  O/オプション  H/ヘルプ
                         HELP: ヘルプの利用法
《H/ヘルプの利用法》  C/目次  《I/索引》  《P/サポート》

 オンラインヘルプは次の方法で利用してください.

   ◆ ヘルプを参照するには, [F･1] を押すか, マウスの右ボタンを押します.
   ◆ BASIC のステートメントの一部, QuickBASIC のメニュー, コマンド, ダイアロ
     グボックスのヘルプを参照するには, カーソルを置いて [F･1] を押します.
   ◆ 画面に表示されたヘルプ情報を消去するには, [ESC] を押します.
   ◆ ヘルプ画面をスクロールするには, [後頁] を押します.

 ヘルプ情報は, 次のどちらかの位置に現れます.
   ◆ 画面のいちばん上のヘルプウィンドウの中
   ◆ 画面中央に現れるダイアログボックスの中

                                Untitled

                               ダイレクト モード
<SHIFT+F1=ヘルプ> <F6=窓切換> <ESC=取消> <CTRL+F1=次へ> <GRPH+F1=前へ>
```

■ヘルプ画面「ヘルプの利用法」

ら、カーソルキーで「H／ヘルプ」を反転表示させて⏎キーを押すか、キーボードから直接「H」キーを押して選びます。

また、ショートカットキーの機能を使って SHIFT ＋ f･1 でも呼び出すことができます。

SHIFT ＋ f･1 キーを押すと、ヘルプ画面のダイアログボックスが表示され、オンラインヘルプの使い方が表示されます。このなかからヘルプの目次と索引に移ることもできます。

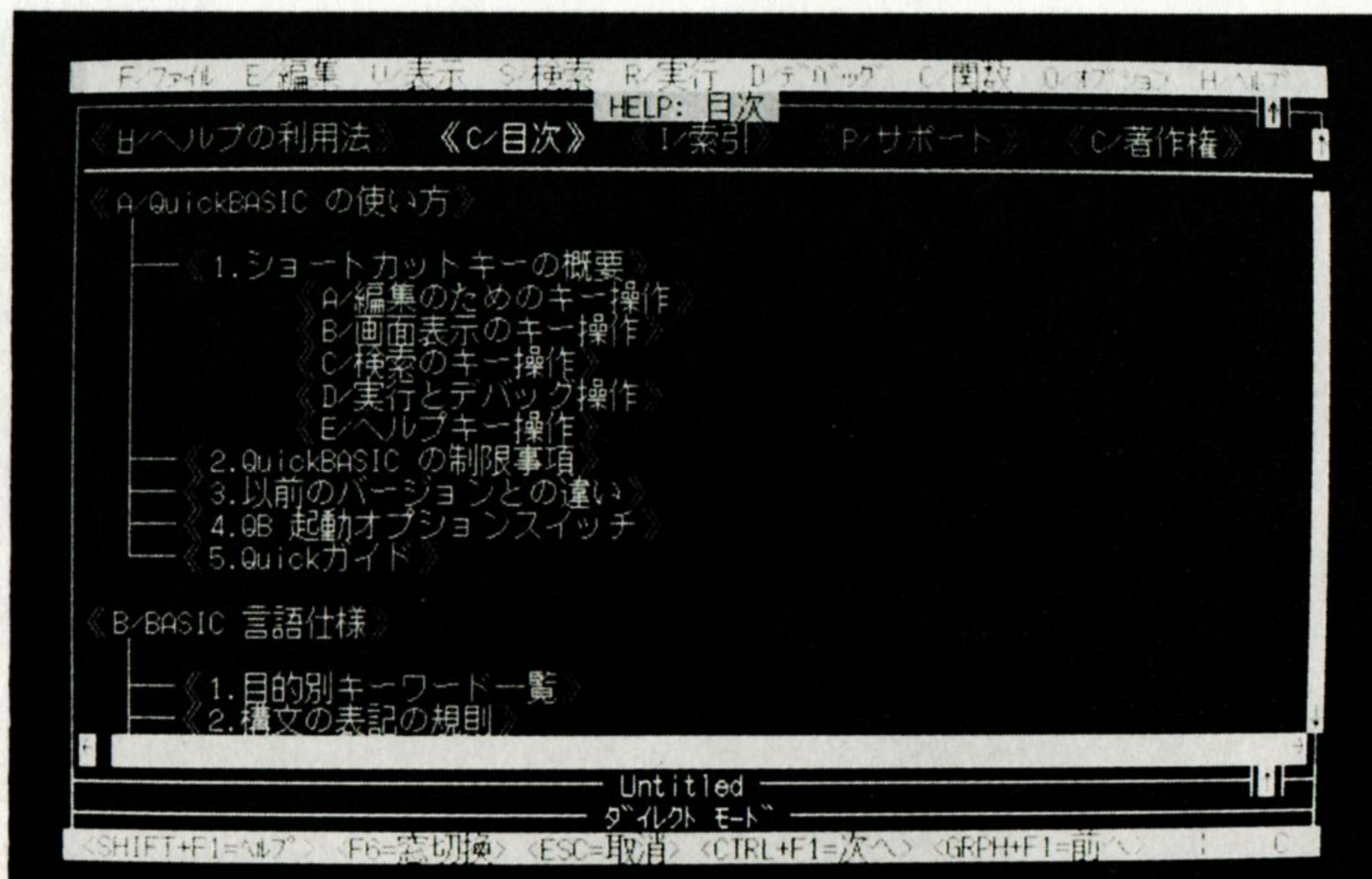
マウスでは、メニューバーから「H／ヘルプ」を左クリックすると、プルダウンメニューが開きます。

プルダウンメニューからは、「C／目次」を選択すると、図のような画面が表示されるので、必要な項目を選んで解説を読みます。

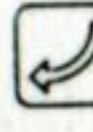
●ヘルプ利用中の操作

「H／ヘルプ」メニューから「C／目次」「I／索引」などのダイアログボックスを開いて、具体的に解説を読みたい項目や、キーワードを選択します。

キーボードでは、 TAB キーで選択可能な項目にカーソルを移動して⏎キーを押すか、項目の先頭のアルファベットや数字を入力して、ヘルプの内容を表示させます。

```
F/ファイル E/編集 V/表示 S/検索 R/実行 D/デバッグ C/関数 O/オプション H/ヘルプ
HELP: 目次
《H/ヘルプの利用法》 《C/目次》 《I/索引》 《P/サポート》 《C/著作権》
《A/QuickBASIC の使い方》
 ├─《1.ショートカットキーの概要》
 │      《A/編集のためのキー操作》
 │      《B/画面表示のキー操作》
 │      《C/検索のキー操作》
 │      《D/実行とデバッグ操作》
 │      《E/ヘルプキー操作》
 ├─《2.QuickBASIC の制限事項》
 ├─《3.以前のバージョンとの違い》
 ├─《4.QB 起動オプションスイッチ》
 └─《5.Quickガイド》
《B/BASIC 言語仕様》
 ├─《1.目的別キーワード一覧》
 ├─《2.構文の表記の規則》
Untitled
ダイレクト モード
<SHIFT+F1=ヘルプ> <F6=窓切換> <ESC=取消> <CTRL+F1=次へ> <GRPH+F1=前へ>
```

■ヘルプ画面「目次」

マウスでは、「H／ヘルプ」のメニューからプルダウンメニューまでの操作は、マウスカーソルを合わせて左クリックしますが、ヘルプのダイアログボックスを開いてからは、マウスカーソルが赤色に変わります。

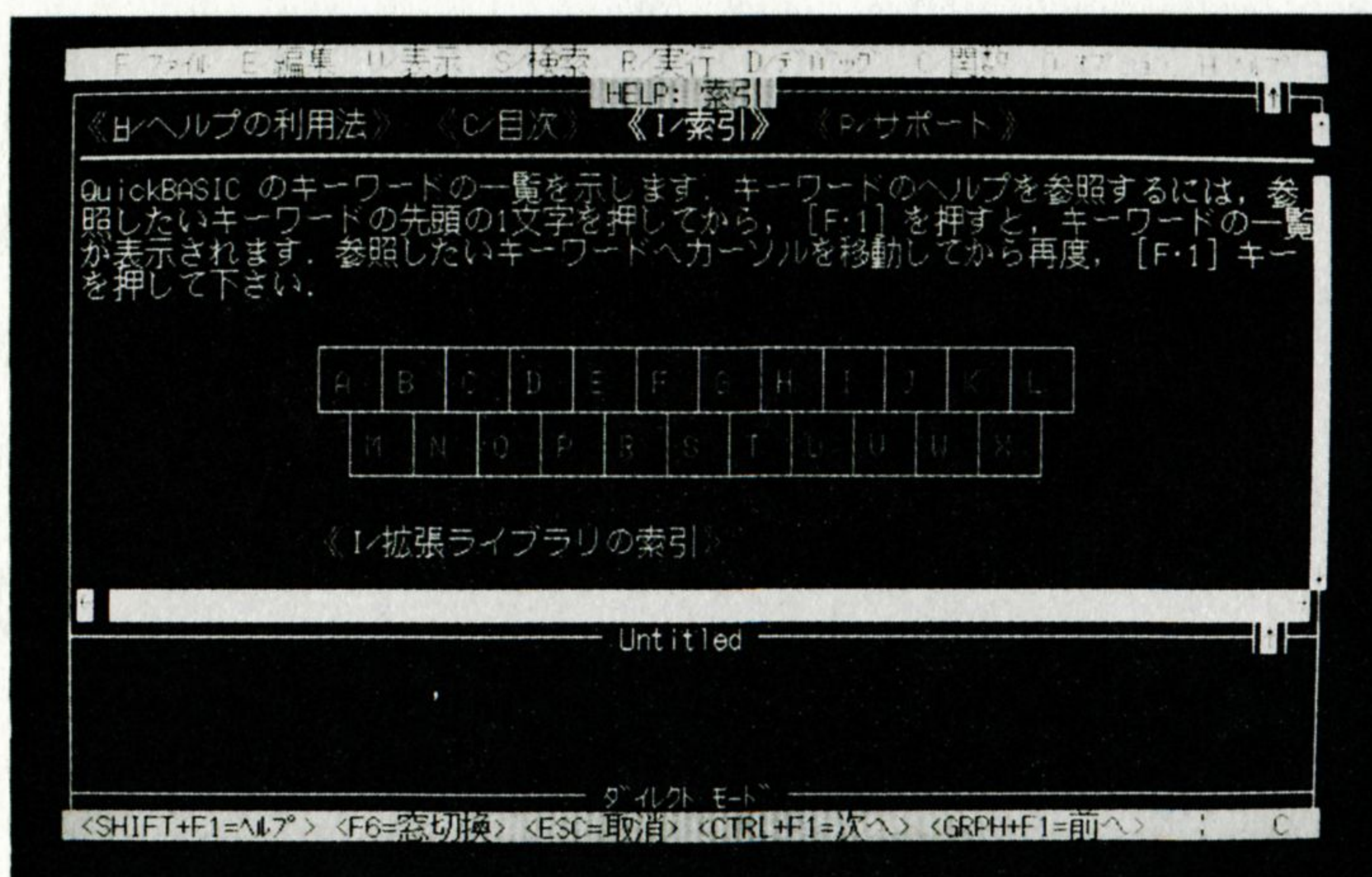
これは、QBアドバイザのなかの操作に移っていることを示していて、QBアドバイザ内では、赤いマウスカーソルをヘルプの項目に合わせて、「右」クリックして解説を選択して表示します。

ヘルプ機能を止めて、Quick BASICのエディタ画面に戻るには、マウスを使っている場合も、キーボードでの場合も、キーボードから ESC キーを押します。

ヘルプメニュー

ヘルプには、キーワードの先頭文字から解説を選んだり、機能ごとに分類された項目から探していったり、プログラミング中にすぐにキーワードのヘルプを見られる機能が配置されています。

先頭１文字から探す　　**I／索引**　索引は、Quick BASICのステートメントや関数などのキーワードを先頭の１文字から選択して表示します。

「I／索引」を選択すると、画面にダイアログボックスが開かれ、その中央にアルファベットの一覧表が表示されます。

キーボードからは、この一覧のなかに TAB キーでカーソルを移動するか、キーボードから直接アルファベットのキーを押してから⏎キーを押すと、アル

■ヘルプ画面「索引」

ファベットで始まるキーワードの一覧が表示されます。このなかから解説を表示させるキーワードを、TABまたはカーソルキーで選択します。

マウスの場合は、赤いマウスカーソルを直接アルファベットの上に移動して右クリックし、キーワードの一覧が表示されたら、そのなかから解説の必要なキーワードをマウスカーソルで選択します。

内容の目次から探す　**C／目次**　目次は、「A／Quick BASICの使い方」と「B／BASIC 言語仕様」に大きく分類されています。

A／Quick BASICの使い方　Quick BASICのショートカットキーやQBアドバイザの使い方などの解説がまとめられています。

B／BASIC 言語仕様　Quick BASICの文法についての解説がまとめられています。「1．目的別のキーワード一覧」は、Quick BASICのステートメントや関数の使い方を機能別に分類して整理しているので、プログラミング中に自分のやりたいことを実現できるキーワードを探し出すのに向いています。

キーワードの解説を直接読む　**T／キーワード**　エディタ画面でプログラミング中の場合、カーソルのある位置のステートメントや関数の解説、プロシージャやユーザー定義変数について表示します。

オンラインヘルプの使い方　**H／ヘルプの利用法**　オンラインヘルプの使い方を表示します。この項目から、「C／目次」や「I／索引」などの項目を選択していくことができます。

ハイパーリンク

Quick BASICのステートメント、関数などのキーワードには「ハイパーリンク」という、関連する項目を次つぎと参照していくことができる機能があります。

ハイパーリンクは、キーワードのヘルプを表示している画面の最上部に「D／詳細」、「E／プログラム例」、キーワードの解説の最後に「参照」として関連するステートメントや関数が示されています。

それぞれ「《　》」の二重かっこでくくられているので、ここにカーソルを合わせて⏎キーを押すか、マウスカーソルを合わせて右クリックすると、それぞれの内容が表示されます。

ヘルプ画面からハイパーリンクの機能を使ったときに、Quick BASICアドバイザディスクがBドライブに入っていないと、

ファイルQB45QCK.HLPが見つかりません

ファイルの入ってるディスクをドライブに差込み実行するか

O／オプション＋P／パス設定でヘルプファイルのパスを変更してください

というダイアログボックスが開かれて注意されます。このとき、Bドライブのフロッピーディスクを Quick BASIC アドバイザディスクに入れ替えて、⏎キーを押すと、ハイパーリンクの機能を使うことができます。

詳しい解説を読む　　**D／詳細**　キーワードヘルプでは、ステートメントや関数の簡単な意味と、パラメータの指定順序などが示されますが、詳細画面では、パラメータの解説やステートメントを使用するうえでの注意などの詳しい内容が表示されます。

プログラムの実例を見る　　**E／プログラム例**　ステートメントや関数を使っ

```
 F/ファイル  E/編集  V/表示  S/検索  R/実行  D/デバッグ  C/関数  O/オプション  H/ヘルプ
                    HELP: PRINT:QuickSCREEN
《QuickSCREEN》 《D/詳細》 《E/プログラム例》 《C/目次》 《I/索引》

PRINT - 画面上にデータを出力するデバイス I/O ステートメントです.

構文
  PRINT [expressionlist][{,|;}]
    ◆ すべての引数を省略すると, 空白行が出力されます.
    ◆ expressionlist を指定すると, その式の値が画面上に出力されます.
    ◆ リスト中の式は, 数式でも文字列式でも構いません (ただし, 文字列式はダブ
      ルクォーテーションマークで囲んでください).
    ◆ オプションの文字 "," と ";" については, 《D/詳細》 を参照してくださ
      い.

PRINT キーワードのその他の使い方
    《PRINT USING》  書式設定したテキストを画面に出力します.
    《PRINT #》  データをシーケンシャルファイルに出力します.
                            Untitled

                          ダイレクト モード
<SHIFT+F1=ヘルプ> <F6=窓切換> <ESC=取消> <CTRL+F1=次へ> <GRPH+F1=前へ>   :   C
```

■ヘルプ画面「キーワードヘルプ」

E／プログラム例
参照
D／詳細　Quick SCREEN

たサンプルプログラムを見ることができます。プログラムの解説や、パラメータ指定方法の実例を示しているので、プログラミングの参考になります。

また、示されたサンプルプログラムは、SHIFT＋カーソルキーで範囲を指定して「E／編集」メニューの「C／コピー」と「P／ペースト」機能を使って、ヘルプ画面から他の画面に複写することができます。

この機能を利用して、エディタ画面で作成中のプログラムにコピーして利用したり、ダイレクトモードでプログラムの動きを試してみることができます。

関連する項目を参照する　**参照**　ヘルプ画面の最下部に関連するステートメントや関数が「《　》」の二重かっこにくくられて示されています。

参照したい項目にカーソルを合わせて⏎するか、マウスカーソルを合わせて右クリックすると、選択されたキーワードのヘルプ画面が表示されます。

元のヘルプ画面に戻る　**Quick SCREEN**　ハイパーリンクを利用して「D／詳細」や「E／プログラム例」を表示しているときに、元のヘルプ画面に戻るためには、「Quick SCREEN」を選択します。

●ハイパーリンクの活用

QBアドバイザのハイパー機能を活用すれば、マニュアルをまったく開かずにプログラミングできる「マニュアルレス」の操作環境が実現できるだけではなく、次つぎと関連項目を調べていくことによって、BASICの勉強にも利用することもできるでしょう。

ハイパーリンクの活用は、利用者のアイデア次第で、どんどん広がる可能性を秘めています。

自分好みのQBに

画面の色やマウスの機能など使いやすく好きずきに

Quick BASICを利用するときの画面の色や、マウスの機能、メニューの詳しさなど、付加的な設定を行うために、「オプション」のメニューが用意されています。

オプションメニュー

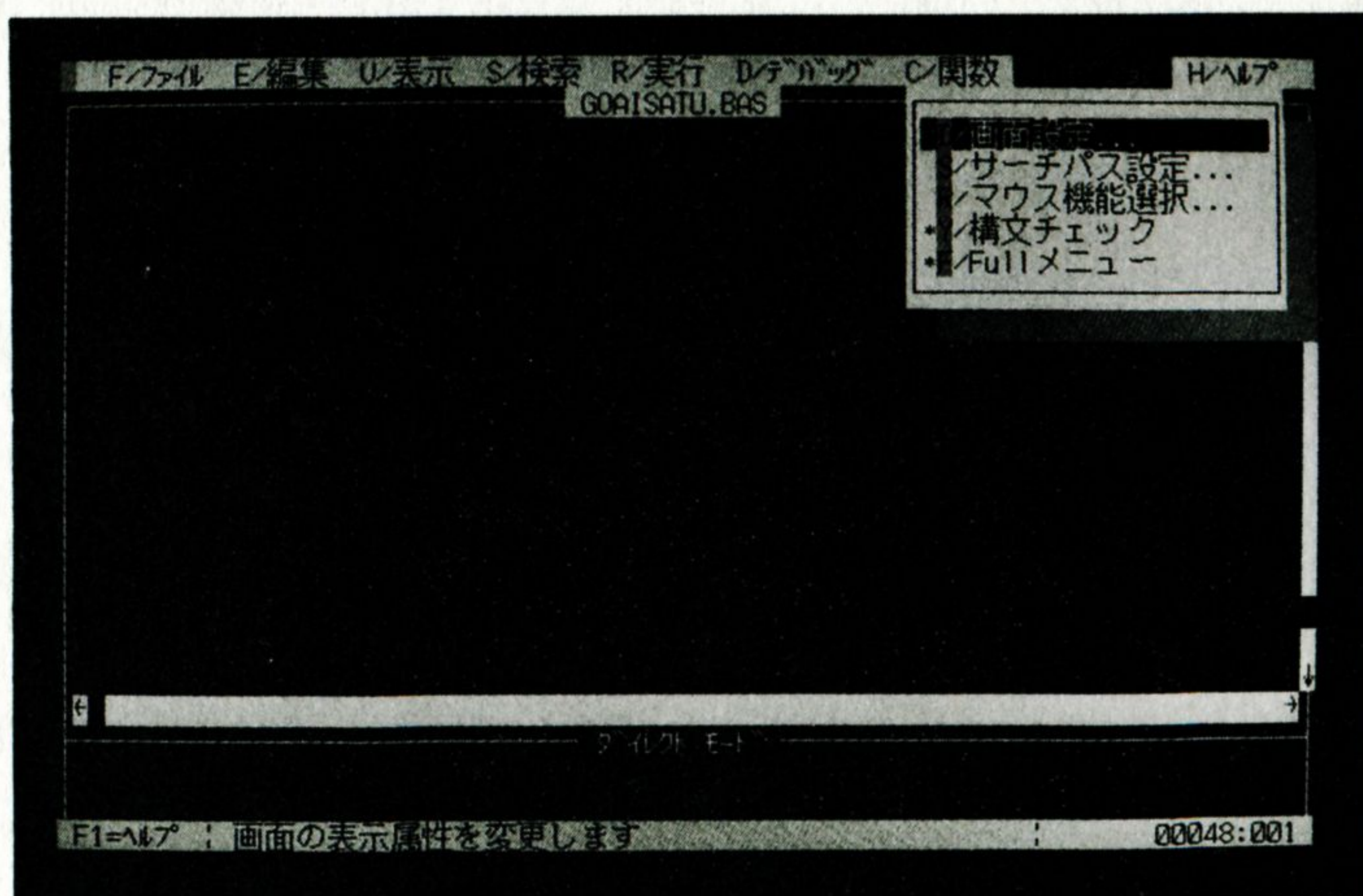
エディタ画面の設定　　**D／画面設定. . .**　このコマンドを実行すると、画面設定のダイアログボックスが現れ、

1．「テキスト」の文字色

2．「現在の実行ライン」の行の色

3．「ブレークポイント」の行の色

を設定することができます。それぞれの色は、「F／文字色」から選択します。

変更したい項目に［TAB］とカーソルキーでカーソルを合わせ、［スペース］バーを押して「＊」（アスタリスク）をつけます。次に、［TAB］キーを押して「F／文字色」のボックスに移り、カーソルキーで色を指定して、［↵］キーを押すと、指定した色に変わります。

■プルダウンメニュー（オプション）

スクロールバーの「表示」「非表示」を切り替えるためには、

S／スクロールバー

の前についている［　］（中かっこ）に、TABキーでカーソルを移動して、スペースバーを押します。「*」（アスタリスク）をつければ「表示」に、消せば「非表示」に設定できます。

エディタ画面でTABキーを押すと、タブ位置ごとにカーソルが移動しますが、このタブで移動する桁数を指定するのが、

T／タブ位置：

です。この右側にカーソルを合わせて、数字キーで指定すると、その数ごとのタブ位置が設定されます。

パスの設定　**S／サーチパス設定...**　Quick BASICの種々の作業に必要なプログラムなどのファイルは、フロッピーディスクのなかにディレクトリをつくって登録されています。ファイルの登録されているドライブとディレクトリを指定するのがこのコマンドです。

コマンドを実行すると、ダイアログボックスが開いて、

X／実行ファイル：

I／インクルードファイル：

L／ライブラリファイル：

E／ヘルプファイル：

のパスの設定を求められます。それぞれの右側に位置しているボックスのなかに、TABキーを使ってカーソルを移動して、キーボードからファイルのあるドライブ名とディレクトリ名を入力します。

普通は、インストールのときにすべて設定されているので、その設定を変更する必要はありません。

マウスの機能を設定　**R／マウス機能選択...**　マウスの右ボタンの機能を設定します。初期設定では、ステートメントや関数などの上にカーソルを合わせて、マウスの右ボタンをクリックすると、キーワードヘルプが表示されるようになっていますが、このコマンドで、この機能を使わずに「クリック行まで実行」の機能に変えることができます。

このコマンドを実行すると、ダイアログボックスが開いて、

C／キーワードヘルプの表示

E／クリック行まで実行

と、マウスの右クリックの機能をどちらに設定するかたずねられます。

設定する方の左側の「(　)」(カッコ) 内にカーソルを移動して「*」(アスタリスク) をつけます。

プログラムの作成中は「キーワードヘルプの表示」を、テストランを行いながらのデバッグ中には「クリック行まで実行」の機能に設定しておくと便利です。

自動構文チェックの設定　**Y／構文チェック**　エディタ画面でプログラムを作成しているときに、入力されたステートメントなどがQuick BASICの文法に合っているかを自動的に調べるのが、構文チェックの機能です。

Y／構文チェックは、切り替え式のスイッチになっています。初期設定では、この機能が働くようにオンになっていて、メニュー内の「Y／構文チェック」の前に「*」(アスタリスク) がついていますが、このコマンドを選択するとオフになって、「*」が消えます。

プログラムではなく、プログラムの使い方などを示すドキュメントファイルをエディタ画面をワープロ代わりに使って書くときには、この機能をオフにしておくとよいでしょう。

メニューの詳しさの選択　**F／Fullメニュー／Easyメニュー**

「F／Fullメニュー／Easyメニュー」は、切り替え式のスイッチになっていて、初期設定では「Easyメニュー」になっています。

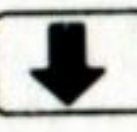
詳しいメニューが必要なときは、このコマンドを選択して「F／Fullメニュー」に切り替え、コマンドの前に「*」(アスタリスク) をつけます。

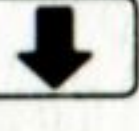
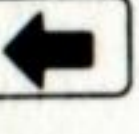
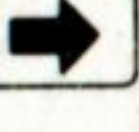
「O／オプション」のメニューで設定した条件は、Quick BASICのプログラムを終了したときに、自動的にフロッピーディスクに登録されます。そのため、次にQuick BASICを起動したときには、前回と同じ設定で利用することができます。

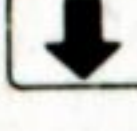
PART○3

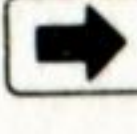
Quick BASICでなにかする

編集する

対話しながら自動構文でも検索でも

エディタの基本操作

Quick BASICをスタートすると、初期画面のビューウィンドウが開かれています。ビューウィンドウは、プログラムの作成、修正などをするエディタになっています。

キーボードから入力されるプログラムは、カーソルの位置に1文字ずつ記入されていきます。

N_{88}-日本語BASIC(86)などの一般的なBASICのように、1行ごとに⏎キーを押して確定していく必要はありません。画面でのリストの並びがそのままプログラムになります。

●カーソルの動かし方

カーソルは、[↑][↓][←][→]だけではなく、次のキーを使って動かすことができます。

[CTRL]+E　　1文字**上**に移動＝[↑]
[CTRL]+X　　1文字**下**に移動＝[↓]
[CTRL]+S　　1文字**左**に移動＝[←]
[CTRL]+D　　1文字**右**に移動＝[→]

それだけではなく、

[CTRL]+W　　1行ずつ**下**にスクロール＝ウィンドウの上端で[↑]
[CTRL]+Z　　1行ずつ**上**にスクロール＝ウィンドウの下端で[↓]
[CTRL]+R　　1画面**下**にスクロール＝[ROLL UP]
[CTRL]+C　　1画面**上**にスクロール＝[ROLL DOWN]
[CTRL]+A　　1語**左**に移動＝[CTRL]+[←]
[CTRL]+F　　1語**右**に移動＝[CTRL]+[→]

の機能も果たします。

この操作法を覚えると、ホームポジションでカーソル操作ができます。

ビューウィンドウ	編集メニュー
カーソル移動	メニューバー
スクロール	プルダウンメニュー

●編集範囲の指定方法

プログラムリストをつくったり、複写や削除の編集作業をするためには、事前に編集する範囲を指定しておかなければなりません。

範囲の指定は、指定する初めの位置にカーソルを置き、

SHIFT + ↑
SHIFT + ↓
SHIFT + ←
SHIFT + →

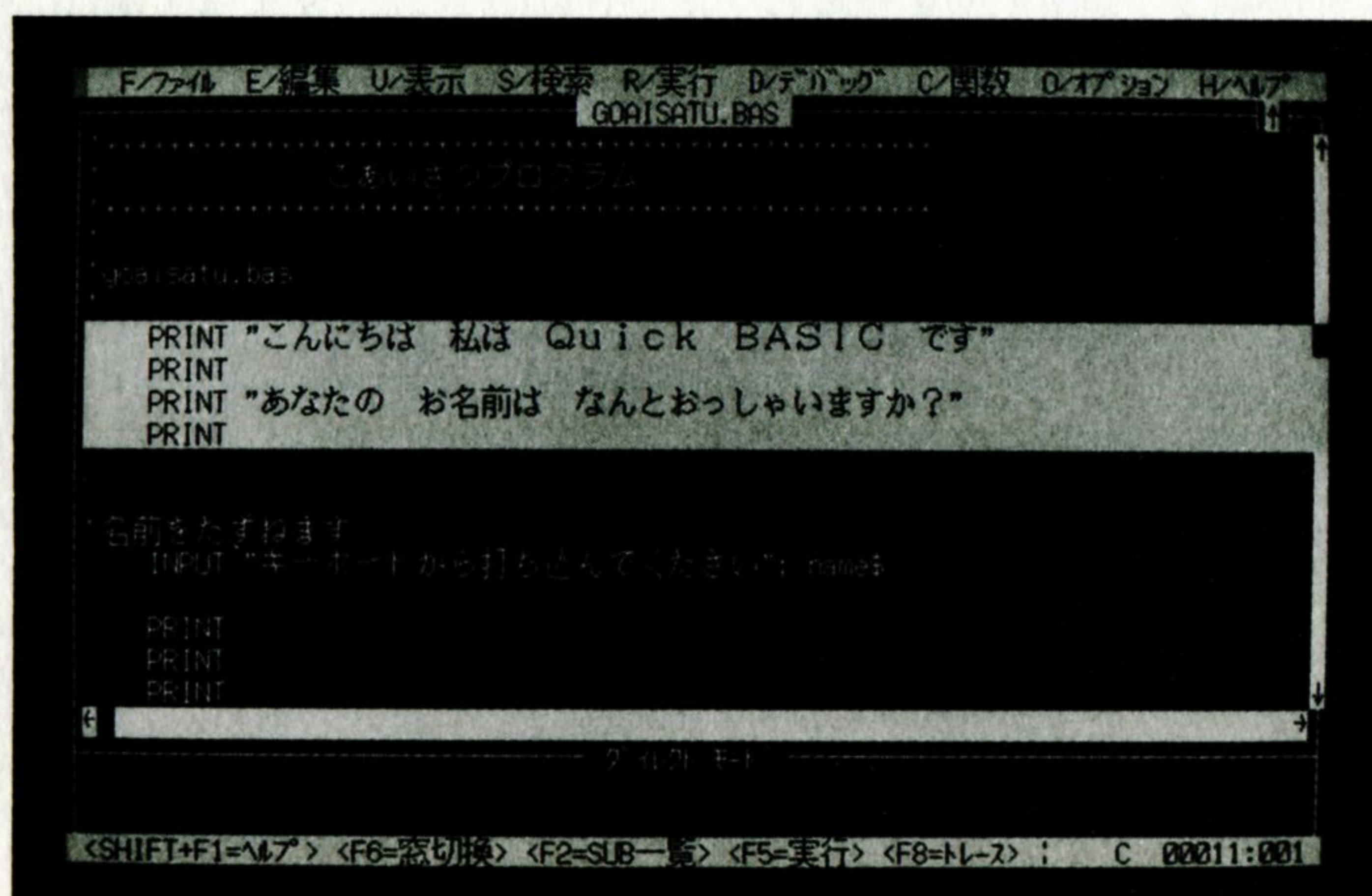
として、カーソルの方向に反転表示の部分を広げて指定します。

指定した範囲を「編集」メニューにあるコマンドを使って編集します。

●コマンドの選択

コマンドを選択するためには、メニューバーのなかからコマンドの配置されているメニューを選んで、プルダウンメニューを開き、コマンドの一覧を表示します。この一覧のなかから必要なコマンドをキーボードまたは、マウスで選択します。

キーボードでは、GRPHを押して、メニューバーをアクティブにして選択可能状態にします。

```
 F/ファイル  E/編集  U/表示  S/検索  R/実行  D/デバッグ  C/関数  O/オプション  H/ヘルプ
                              GOAISATU.BAS
 ''''''''''''''''''''''''''''''''''''''''''''''
 '              ごあいさつプログラム
 ''''''''''''''''''''''''''''''''''''''''''''''

 'goaisatu.bas

    PRINT "こんにちは  私は  Quick  BASIC  です"
    PRINT
    PRINT "あなたの  お名前は  なんとおっしゃいますか?"
    PRINT

 '名前をたずねます
    INPUT "キーボードから打ち込んでください"; name$

    PRINT
    PRINT
    PRINT

 <SHIFT+F1=ヘルプ> <F6=窓切換> <F2=SUB一覧> <F5=実行> <F8=トレース>  |  C  00011:001
```

■編集範囲の指定

メニューバーのなかの必要なコマンドが配置されているメニューを、カーソルキー［←］［→］でアクティブにして、［↵］キーで選択するか、直接メニューに設定されたアルファベットの文字を入力します。

すると、選択されたメニューの下にプルダウンメニューが開き、コマンド一覧が表示されます。カーソルキー［↑］［↓］で、実行したいコマンドをアクティブにして、［↵］キーで選択するか、直接コマンドに設定されたアルファベットの文字を入力します。

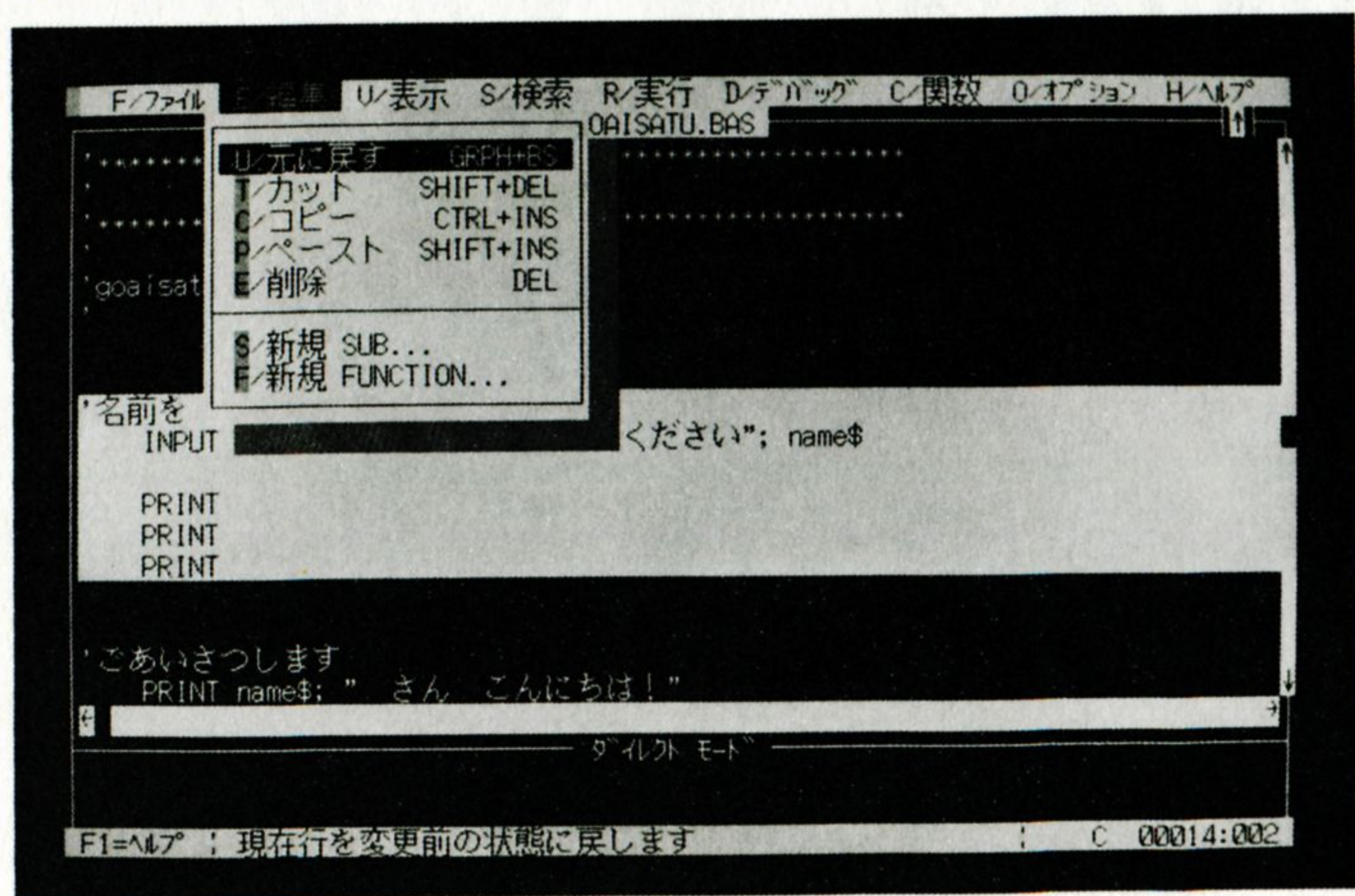
マウスでは、メニューにマウスカーソルを移動して、左クリックして、プルダウンメニューを開き、つづけて実行したいコマンドにマウスカーソルを移動して左クリックします。これで、選択されたコマンドが実行されます。

●ダイアログボックスの操作

Quick BASICのコマンドを選択すると、より細かな設定が必要な場合に「ダイアログボックス」が開いて各種の設定を求められます。ダイアログボックスは対話のためのボックスで、キーボードでもマウスでも操作できます。

ダイアログボックスのなかには、いくつかのボックスが配置されています。このボックスにマウスカーソルやカーソルを移動して、各種の設定や選択、データの入力を行います。

マウスでは、ボックス内にマウスカーソルを移動させると、反転表示されたり、マークがついたりして、アクティブな状態になります。左クリックして設定、選択します。

```
 F/ファイル  E/編集  V/表示  S/検索  R/実行  D/デバッグ  C/関数  O/オプション  H/ヘルプ
                                 OAISATU.BAS
 '.......  U/元に戻す     GRPH+BS  ....................
 '         T/カット     SHIFT+DEL
 '.......  C/コピー      CTRL+INS  ....................
 '         P/ペースト   SHIFT+INS
 'goaisat  E/削除             DEL
 '
           S/新規 SUB...
           F/新規 FUNCTION...
 '名前を
     INPUT                         ください"; name$

     PRINT
     PRINT
     PRINT

 'ごあいさつします
     PRINT name$; " さん こんにちは！"

 ─────────────────── ダイレクト モード ───────────────────

 F1=ヘルプ ： 現在行を変更前の状態に戻します          ：  C  00014:002
```

■プルダウンメニュー画面

編集する

ボックス　反転表示

マーク　確認

プルダウンメニュー　アクティブ　取消

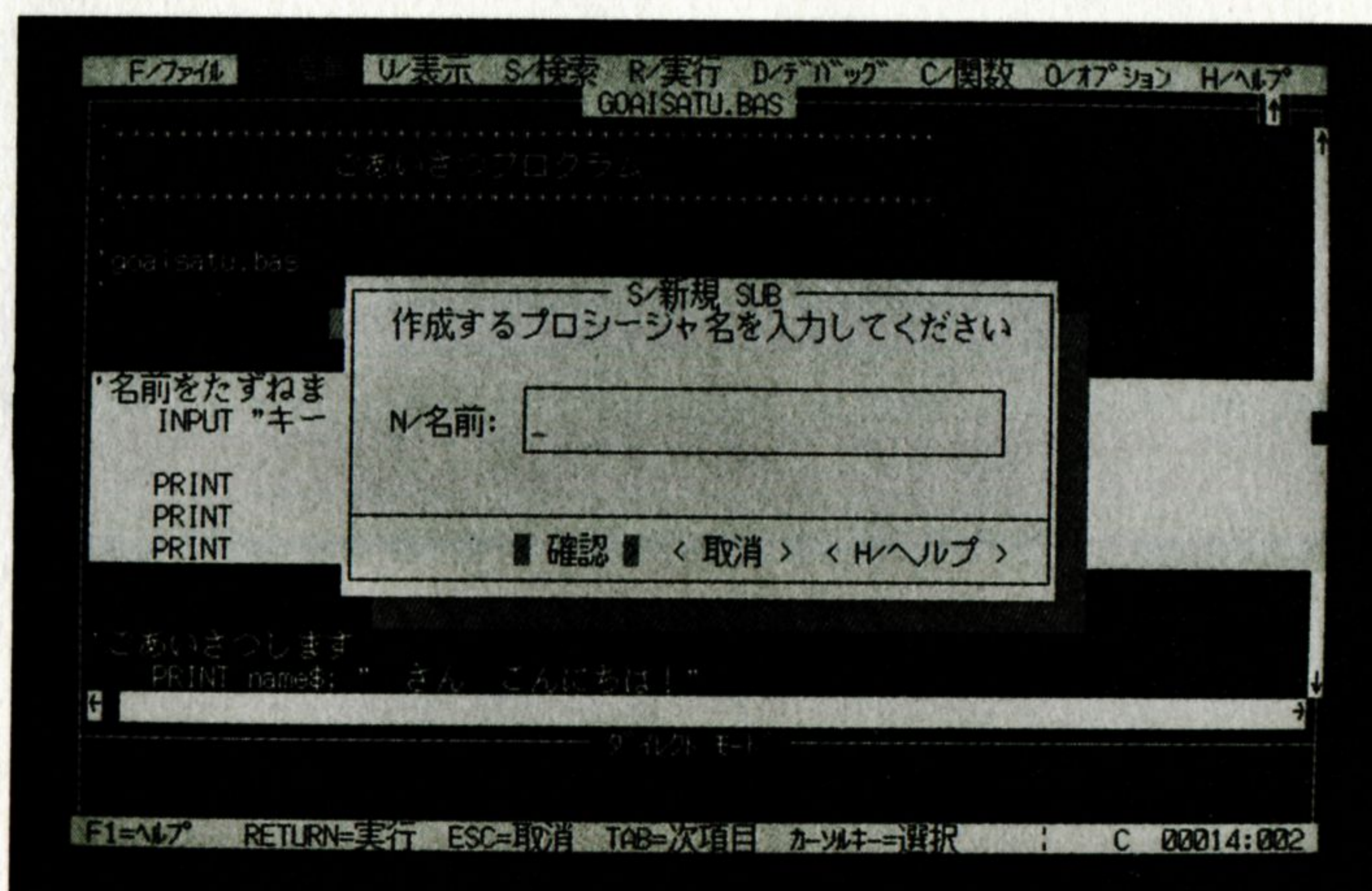
キーボードからは、TABキーで各ボックスにカーソルを移動することができ、カーソルキー↑↓←→とスペースバーでアクティブ状態やマークを設定し、⏎キーを押して設定、選択します。

すべての設定が終わったら、「**確認**」のボックスをマウスまたはTABキーでアクティブにして、左クリック、または⏎キーを押すと、設定された内容でコマンドが実行されます。

設定を取り消したい場合は、「**取消**」のボックスを選択します。

●自動フォーマット

Quick BASICのエディタでプログラムを作成すると、1行入力するごとに、プログラムのフォーマットを次のように、Quick BASICの標準的な形式に自動的に整えます。

BASICのステートメントを大文字にする
変数名、プロシージャ名の大文字、小文字を統一する
演算子の前後にスペースを挿入する
PRINT文の区切りを入れる
不要なスペースを削除する

■ダイアログボックスの画面

編集メニュー

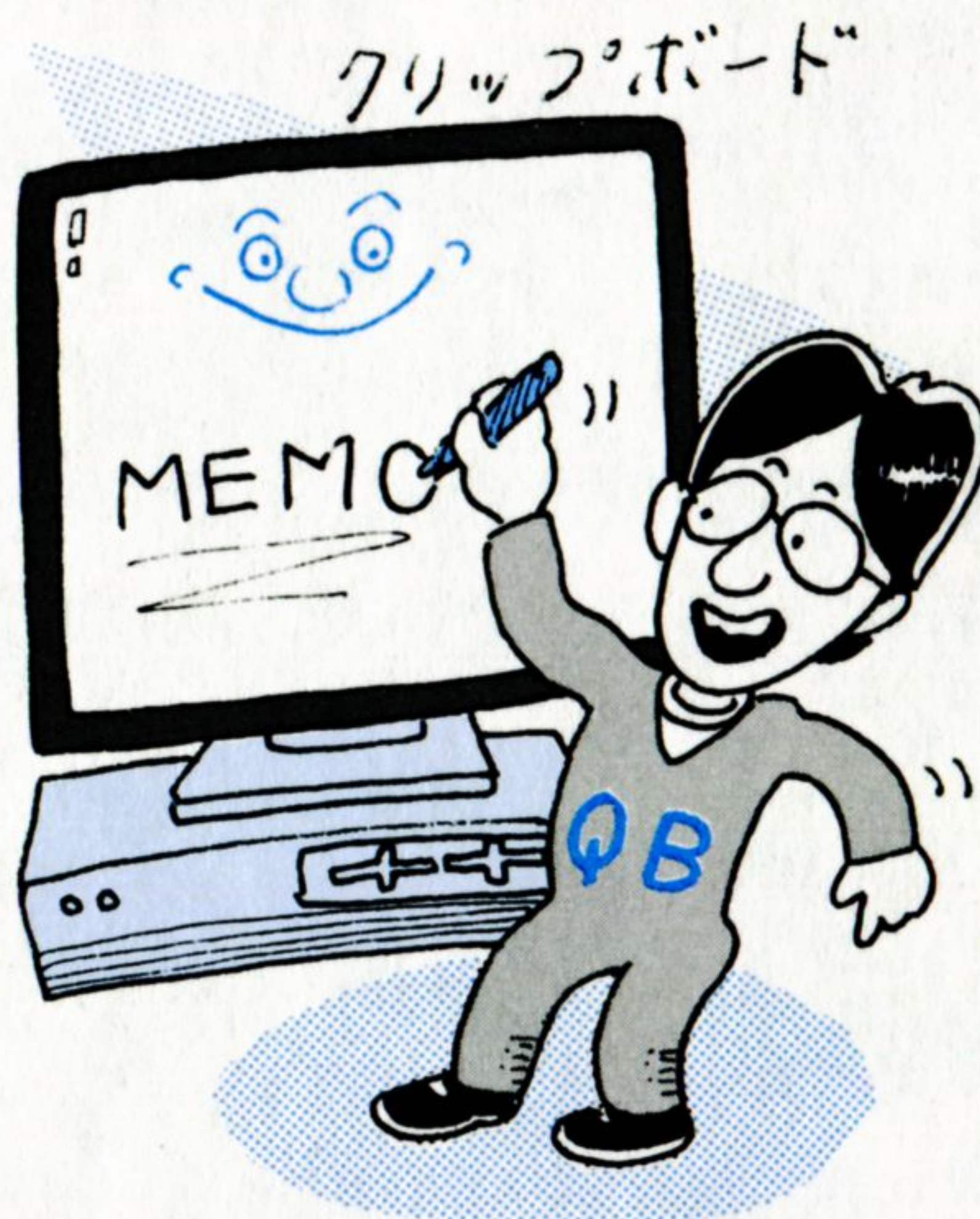
Quick BASIC では、プログラムの編集操作をするコマンドを「E／編集」メニューに配置しています。

編集操作では、「クリップボード」を使って効率的に複写や移動ができます。クリップボードについて初めに触れておきます。

クリップボードは、Quick BASICが管理するメモのようなもので、削除された文字列が一時的に保存されています。

あくまでも一時的なメモですので、次になにか削除したりして、新しくクリップボードに文字列が移されると、以前のものは消えてしまいます。クリップボードには、最後に削除されたものだけが残っているわけです。

このクリップボードとペーストを組み合わせて利用すると、編集機能を便利に使うことができます。

「E／編集」メニューに配置されたコマンドを見ていきましょう。

```
F/ファイル E/編集 V/表示 S/検索 R/実行 D/デバッグ C/関数 O/オプション H/ヘルプ
                        MEISISTR.BAS
TYPE meisi
     Simei AS STRING * 20
     KanjiSimei AS STRING * 20
     Kaisha AS STRING * 40
     Shozoku AS STRING * 40
     Yakushoku AS STRING * 40
     Tel AS STRING * 15
     Fax AS STRING * 15
     Tekiyou AS STRING * 80
END TYPE
DIM SHARED MeisiData AS meisi

OPEN "meisistr.dat" FOR RANDOM AS #1 LEN = LEN(MeisiData)
SaidaiBangou = LOF(1) \ LEN(MeisiData)
CLOSE

DATA.SETTEI MenuKosuu, KoumokuSuu

                        ダイレクト モード
<SHIFT+F1=ヘルプ> <F6=窓切換> <F2=SUB一覧> <F5=実行> <F8=トレース> ¦   C  00054:001
```

■プログラムのオートフォーマット

復活（アンドゥ）　**U／元に戻す**　直前に行った編集作業を取り消します。文字を削除していて、まちがって多く削除しすぎたことに気がついた場合などに操作を取り消して、もとに戻すことができます。

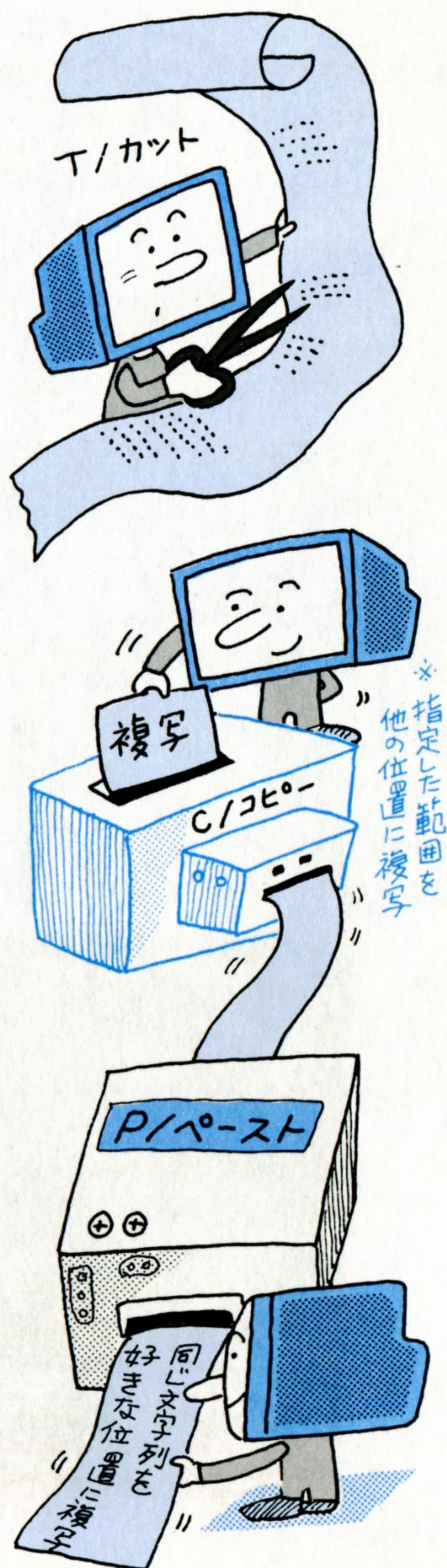
次に正確に削除すれば、また打ち直しをしなくてもすむので便利です。

カット　**T／カット**　指定した範囲を削除します。範囲を指定してこのコマンドを選択すると、反転表示された部分が削除されて、「クリップボード」に移されます。後ろにつづく文字列がある場合は、左へ詰められます。

複写（コピー）　**C／コピー**　指定した範囲を他の位置に複写します。範囲を指定してこのコマンドを選択すると、画面上は文字列を残して、反転表示された部分がクリップボードに移されます。

次に複写する位置で左クリックするか、カーソルを移動して、⏎キーを押すと、指定された文字列がコピーされます。

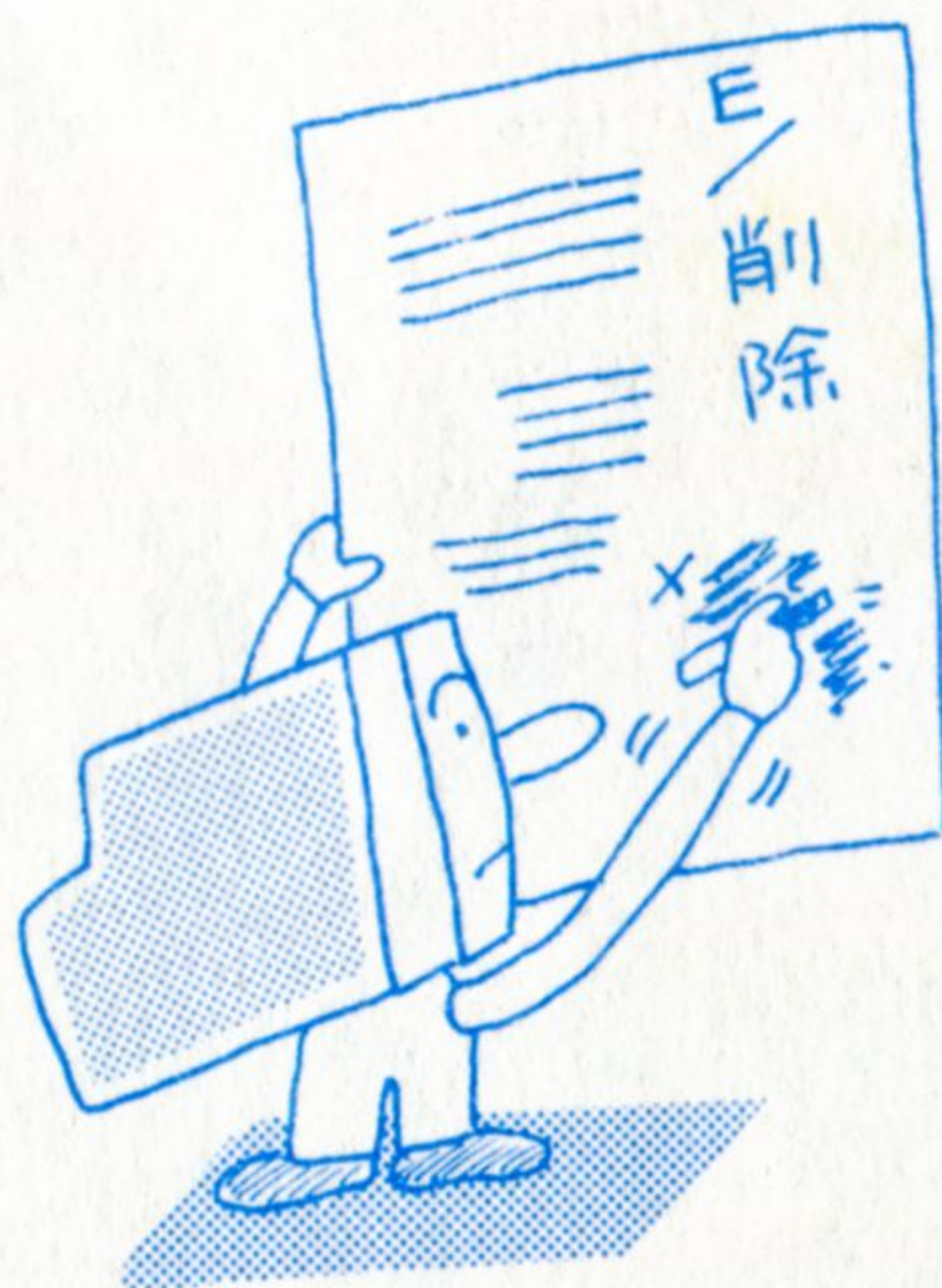
クリップボードの文字列を「P／ペースト」コマンドを利用して、複数の場所にコピーすることもできます。

貼付（ペースト）　**P／ペースト**　クリップボードの文字列を複写します。直前に削除や複写された文字列は、クリップボードに保存されていますが、このコマンドが選択されると、保存されている文字列をカーソルのある位置から複写します。

ペーストコマンドを使えば、カットと組み合わせてコピーの代わりにすることもできます。同じ文字列をいくつでも、好きな位置に複写することができます。

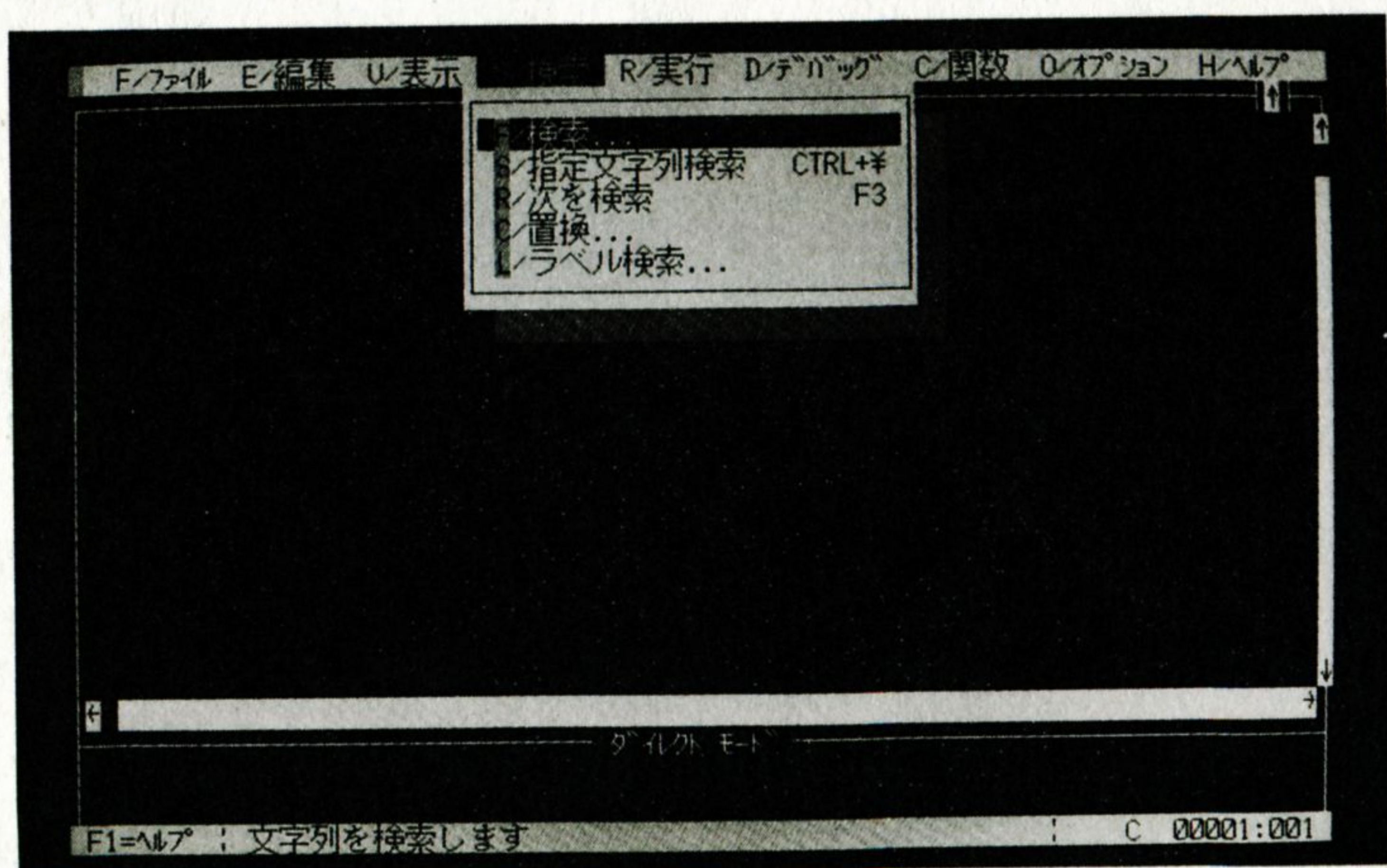
削除

E／削除　指定した範囲を消去します。範囲を指定してこのコマンドを選択すると、反転表示された部分が消去されます。

このコマンドは、カットと違って、消した文字列をクリップボードに移さないので、ペースト機能を使うことができません。

まちがえて消去してしまった場合は、「U／元に戻す」で、消してしまった部分を復活させることができます。

検索メニュー

何十行も何百行もあるプログラムをつくっていると、当然ひとつの画面にリストを表示させることができません。

大きなプログラムのなかから特定の文字列がどこにあるのかを探そうとしたときに、すべてのリストを目で追っていたのではたいへんですし、見落としも出てきます。

■プルダウンメニュー（検索）

	R／次を検索	C／置換	V／逐次確認
置き換え	F／検索	F／置換前	C／一括置換
F／検索文字列	S／指定文字列検索	T／置換後	L／ラベル

また、変数名やメッセージの内容などを変えなければならなくなったときなどは、もっとたいへんです。

このようなときに、検索や置き換えの機能を使うと、早く正確にプログラムの作成や変更ができます。

検索　**F／検索...**　編集中のプログラムのなかから、指定した文字列を探し出して反転表示します。

このコマンドを選択すると、ダイアログボックスが開いて、検索する文字列を聞いてきます。

このとき、カーソルが文字列の上にあると、この文字列が「F／検索文字列」のボックスにセットされています。

この文字列を検索する場合は、このまま⏎キーを押すと検索を始めます。

ダイアログボックスのなかの指定は次のとおりです。

検索する文字列

F／検索文字列	検索する文字列を設定する
（検索する範囲）	
1／アクティブウィンドウ	現在アクティブになっているウィンドウのみ
2／カレントウィンドウ	現在ビューウィンドウで編集中のモジュールのみ
3／全体	ロードされているすべてのモジュール
（検索文字の制限）	
M／大小文字の区別	被検索文字列の大文字と小文字を区別する
W／全体一致	被検索文字列の全体が一致した場合のみ検索

指定文字の検索　**S／指定文字列検索**　エディタ画面であらかじめ指定した範囲の文字列を検索します。

あらかじめ検索したい文字列をエディタ画面で反転表示にしてから、このコマンドを選択すると、すぐに検索を始めます。ただし、検索する文字列を複数の行にまたがって指定することはできません。

検索する文字列があらかじめ指定されていないと、「F／検索」と同じ働きになり、ダイアログボックスが開かれます。

検索の続行　　**R／次を検索**　直前に実行した検索文字列をつづけて検索します。

直前に実行した「F／検索」や「S／指定文字列検索」で設定されている、ダイアログボックス内の検索文字列をつづけて検索します。

f・3 にショートカットキーが設定されていますので、いちいちメニューを開いて選択するよりこちらを利用したほうが便利でしょう。

置き換え　　**C／置換...**　文字列を検索して指定した文字列に置き換えます。

このコマンドを選択すると、ダイアログボックスが開いて、検索する文字列と、この文字と置き換えられる文字列を聞いてきます。

ダイアログボックス内で検索と異なる指定は次のとおりです。

F／置換前　　置換される文字列（検索する文字列）

T／置換後　　置換する文字列（置き換わる文字列）

V／逐次確認　　一回ごとに置換するかどうか確認する

C／置換する＝↵

S／置換しない

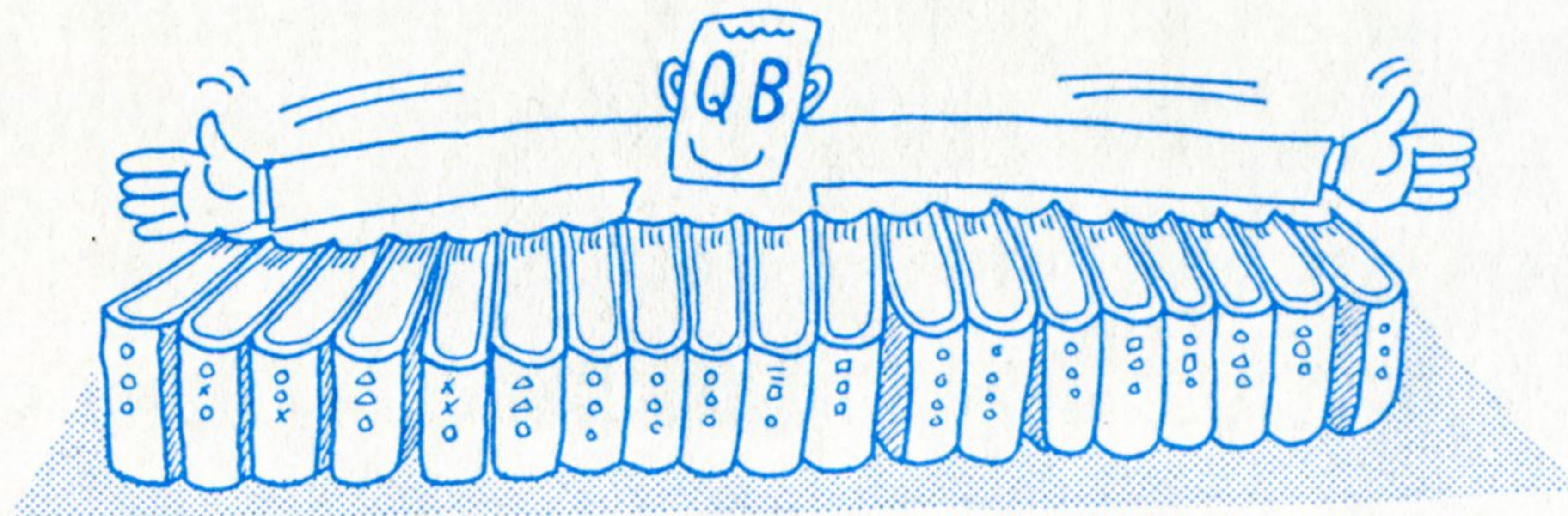
取消　　＝ESC

C／一括置換　　確認せずにすべてを一括して置き換える

ラベルの検索　　**L／ラベル...**　指定したラベルを検索して反転表示します。

このコマンドを選択すると、ダイアログボックスが開いて、検索するラベルの名まえを聞いてきます。

「F／検索文字列」のボックスにセットされている文字列に「：」（コロン）をつけたラベル名を検索します。

ダイアログボックス内の指定は「F／検索...」と同じです。

テストする

入力したプログラムが動くかどうかのテストラン

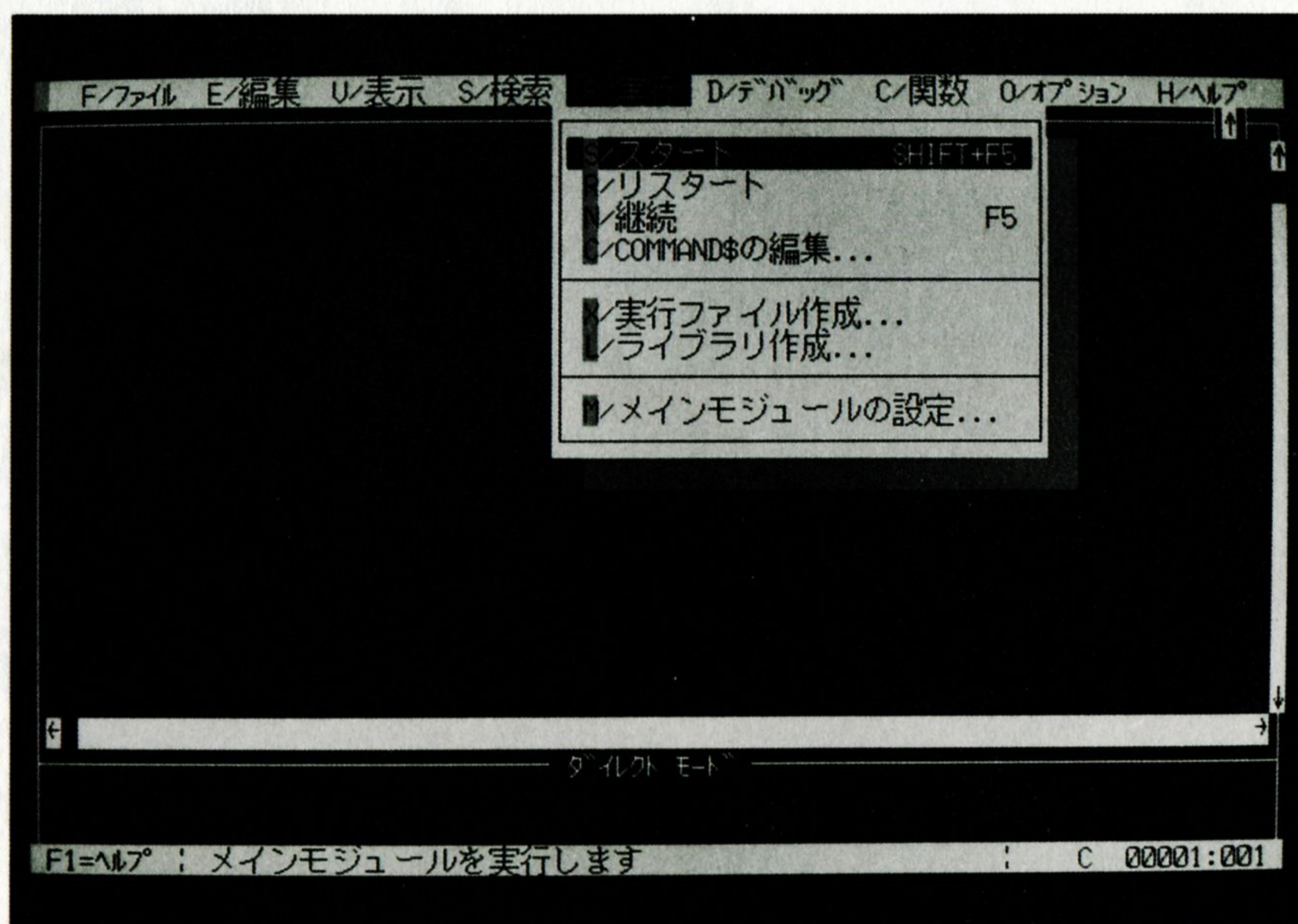
プログラムの作成

いよいよQuick BASICの編集機能を使って、プログラムができあがりました。

Quick BASICでは、プログラムはひとまとまりのしごとをする単位であるいくつかの「モジュール」で構成されています。

Quick BASIC起動時の、初期画面であるエディタ画面に、初めてプログラムを書き始めると、これがメインモジュールになります。

入力し始めは、プログラムの名まえがついていないので、ビューウィンドウの上にあるタイトルバーには、なにも名まえがついていないという意味の「Untitled」が表示されていますが、プログラム名は保存のときにつけます。

```
F/ファイル  E/編集  V/表示  S/検索  R/実行  D/デバッグ  C/関数  O/オプション  H/ヘルプ
                                  S/スタート          SHIFT+F5
                                  R/リスタート
                                  N/継続                    F5
                                  C/COMMAND$の編集...
                                  X/実行ファイル作成...
                                  L/ライブラリ作成...
                                  M/メインモジュールの設定...
                                ダイレクト モード
F1=ヘルプ ｜ メインモジュールを実行します          ｜  C  00001:001
```

■プルダウンメニュー（実行）

実行メニュー

入力したプログラムが、思ったとおりに動くかどうか試してみることをテストランといいます。Quick BASICの環境下でプログラムを作成しているときは、いつでもテストランをすることができます。

テストランのためのコマンドが「R／実行」のメニューに配置されています。

実行　　**S／スタート**　プログラムを初めから実行します。

Quick BASICのインタプリタで、プログラムの最初から、ステートメントをひとつずつ実行していきます。

構文チェックでは検出できないエラーや、構文としては合っているが、実行できないエラーを発見して、プログラムがストップしたりします。

また、プログラムを作成しているときは思ってもみなかった動作や、まちがった考え方をした部分を見つけ出すことができます。

再実行　　**R／リスタート**　停止したプログラムを再度初めから実行します。

STOPキーやブレークポイントなどのデバッグ機能を使って一時停止したプログラムを、再度最初から実行し直します。

続行　　**N／続行**　停止したプログラムをつづけて実行します。

STOPキーやブレークポイントなどのデバッグ機能を使って一時停止したプログラムが、再度停止したステートメントの位置から実行されます。プログラムの停止中にプログラムを修正した場合、実行をつづけられなくなる場合があります。

このとき、ダイアログボックスが開いてQuick BASICから「編集の後、継続ができませんがよろしいですか？」と確認してきますから、「確認」か「取消」を選択します。

コマンドライン引数の入力　　**C／COMMAND$の編集...**　MS-DOSコマンドラインからの引数の入力を設定します。

独立型のプログラムの場合、プログラムを実行する際にMS-DOSのコマンドライン※で、プログラムに必要な引数※を与えて実行する場合があります。

このようなプログラムをQuick BASICの環境下でテストランする場合は、実際にはコマンドラインからの実行ができないため、このコマンドを選択して引数を設定します。

ダイアログボックスが開いて、設定する文字列の入力を求めてきますから、

S／スタート　C／COMMAND$の編集　COMMAND$関数
R／リスタート　コマンドライン　M／メインモジュールの設定
N／続行　引数　ダイレクトモードウィンドウ

キーボードから入力すると、COMMAND$関数で返される文字列として実行が開始されます。

メインモジュールの設定　**M／メインモジュールの設定...**　複数モジュールプログラムでメインモジュールを変更します。

大きなプログラムで、マルチモジュールプログラムになっている場合、プログラムの実行は常にメインモジュールから開始されます。

プログラムの実行を開始するモジュールを変更する必要がある場合には、このコマンドを使ってメインモジュールを変更することができます。

ダイレクトモードの活用

プログラムの作成やテストランを行っていると、ステートメントや関数の働きを試したり、実行結果の表示位置などを試行錯誤で調整したくなる場合があります。

このようなときには、ビューウィンドウの下のダイレクトモードのウィンドウを使って試してみることができます。作成中のプログラムとは別に、直接実行することができる便利な機能です。

●ステートメントのテスト

ダイレクトモードにステートメントを入力して、⏎キーを押せば、結果を試してみることができます。

例えば、ダイレクトモードのウィンドウにカーソルを移して、画面のクリア

※コマンドライン　MS-DOSのプロンプト（「A>」など）につづいて、キーボードから入力される文字列。MS-DOSの内部、外部コマンドにつづいて、コマンド実行に必要な引数が入力されます。

Quick BASICで作成された独立実行型のプログラムも、MS-DOSの外部コマンドとして実行可能ですから、コマンド名であるプログラムのファイル名につづいて、引数を指定してプログラムに引き渡すことができます。

※引数　ステートメントや関数を実行する際に引き渡される値。ステートメントや関数を実行する場合、実行に際して具体的な値を指定する必要のあるものがあります。具体的な数値や文字列がステートメントや関数に引き渡され、それに基づいた処理がされて結果が返されます。

引き渡される具体的な値が引数、引き渡しに使われる仮の変数名などをパラメータといいます。

のテストをしてみましょう。キーボードから、

CLS⏎

とすると、実行画面に切り替わって、画面がクリアされます。

CLS ステートメントのパラメータを変えてみて、反応を実験してみることができます。

●表示テスト

プログラムを作成するビューウィンドウがあるエディタ画面と、実行画面がまったく別になっていますから、プログラムを実行した結果の表示が、プログラムリストといっしょになってしまって、画面が乱れることはありません。

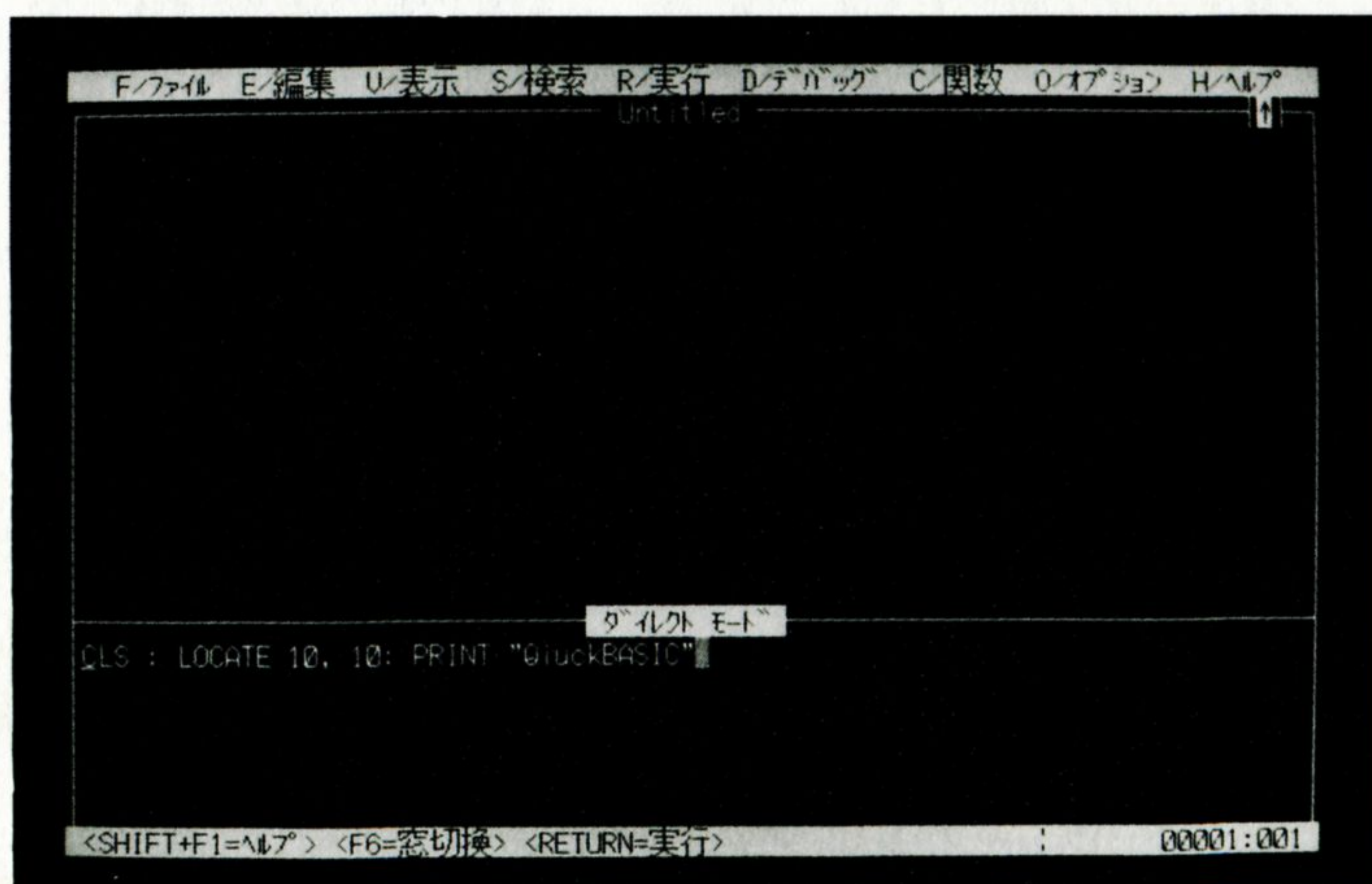
プログラムの一部をダイレクトモードで実行して、思いどおりの画面ができるまで試してみることができます。例えば、LOCATE 文のパラメータをいろいろと変えてみて試してみるのには、

CLS : LOCATE　1Ø, 1Ø : PRINT "QuickBASIC"⏎

として、実行画面で確認してからエディタ画面に戻って、ダイレクトモードで「LOCATE」ステートメントの引数を変更して、何回でもテストすることができます。このように気軽に試行錯誤しながらプログラムを作成できるのも Quick BASIC の特徴のひとつです。

●関数のテスト

関数の働きを確かめることも簡単にできます。ダイレクトモードで MID$の働きを調べてみましょう。

```
 F/ﾌｧｲﾙ  E/編集  V/表示  S/検索  R/実行  D/ﾃﾞﾊﾞｯｸﾞ  C/関数  O/ｵﾌﾟｼｮﾝ  H/ﾍﾙﾌﾟ
                              Untitled

                           ﾀﾞｲﾚｸﾄ ﾓｰﾄﾞ
CLS : LOCATE 10, 10: PRINT "QuickBASIC"

<SHIFT+F1=ﾍﾙﾌﾟ> <F6=窓切換> <RETURN=実行>                    00001:001
```

ダイレクトモード

```
PRINT  MID$ (" QuickBASIC ", 6 , 3)⏎
```

とすると、実行画面に結果が表示されます。

これもかっこ内の引数をいろいろと変えて試すと、変数の機能がよくわかり、プログラムのなかでの使い方をまちがえないですみます。

●変数値の変更

プログラムをテストランしていると、思ったとおりに動作しないことがあります。このようなときは、デバッグ機能を使って、考え違いをしている部分がないかどうか調べていきますが、プログラムを見ているだけではなかなかわかりません。

原因と思われる変数の値を調べるのは、「デバッグ」メニューでできますが、ダイレクトモードでは、変数の値を直接代入して変更し、プログラムを実行してみます。

例えば、ダイレクトモードで、

```
A=10000⏎
```

などとしてプログラムを続行してみます。

こうすると、限界と考えていた値の範囲外で予期しなかった動作をすることが発見されたりします。

●エラーシミュレート

Quick BASICのイベントトラップのうち、エラートラップを使っている場合などは、エラーシミュレートを行って、実際にエラーが起った場合の動作を確かめてみることができます。

```
ERROR  53⏎
```

などとすると、フロッピーディスク上に指定したファイルがない場合のエラーの発生をシミュレートして、エラールーチンの働きが正常かどうか試してみることができます。

ファイルを管理する

プログラムにはまず名まえをつけて

プログラムファイルの保存

プログラムを作成しても、これをフロッピーディスクに保存することができなければ、1回ごとに改めてキーボードから打ち込まなくてはならなくなってしまい、コンピュータを使うメリットはなくなってしまいます。

ファイルの保存方法にもいくつかあります。

標準ファイルは、Quick BASICのエディタでは、プログラムとして読み出したり、訂正したりすることができますが、MS-DOSのTYPEコマンドで表示させたり、ほかのエディタや、ワープロなどで読み出しても、私たちにはプログラムとしては読むことができません。

しかし、標準ファイルはQuick BASICの内部コードで保存されるために、テキストファイルにくらべて小さなファイルになります。

MS-DOSを活用することを考えると、他のソフトとデータのやり取りができるテキスト形式で保存しておくとよいでしょう。

ファイル名は、アルファベット8文字と、「.」（ピリオド）をはさんで拡張子3文字を使ってつけます。

ファイル名には内容を表わす名称や略称などをつけ、拡張子にはファイルの種類を示す略称をつけます。

拡張子には一般的に、

.BAS	BASICのプログラム
.EXE	MS-DOSで直接実行可能なコンパイル後のプログラム
.DAT	プログラムで参照するデータファイル
.DOC	プログラムの使い方などを記した文書ファイル

などを使います。

ファイル　N／新規
名拡張子　Untitled
F／ファイル　O／読込

ファイル名には全角の漢字を使うこともできますが、全角で4文字以上になったり、異常なファイル名をつけると、ファイルを読み出せなくなる場合がありますので、アルファベットと記号だけを使うのがよいでしょう。

ファイルメニュー

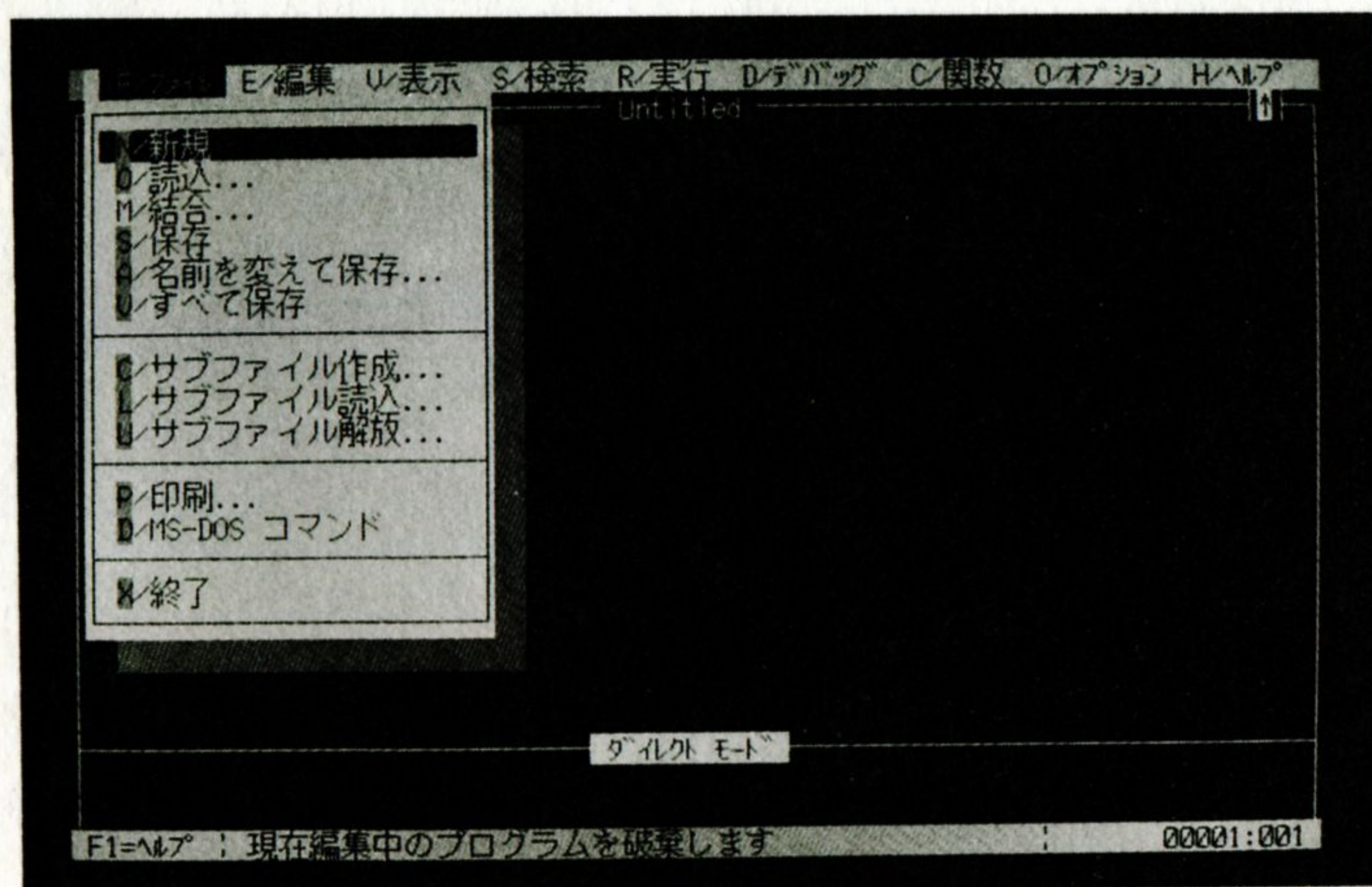
Quick BASICでは、プログラムをファイルとして保存するためのコマンドが「F／ファイル」メニューに配置されています。

ファイルの保存だけではなく、読み出し、印刷、サブファイルの操作、Quick BASICからMS-DOSのコマンドを実行することなどができます。

新しいプログラムの作成　**N／新規**　新しいプログラムを作成します。

すでに読み込まれて編集中であったり、作成中のプログラムがある場合は、消去されて、初期画面のようになにも書かれていないエディタ画面になり、新しいプログラムを作成、編集できるようになります。

タイトルバーもファイル名のない状態「Untitled」に変わりますから、作成したプログラムには保存のときに新たにファイル名をつけます。

プログラムの読み込み　**O／読込...**　フロッピーディスクからパソコンにプログラムを読み出します。

■プルダウンメニュー（ファイル）

このコマンドを選択すると、ダイアログボックスが開いて、読み出すプログラムのファイル名を聞いてきます。

N／ファイル名　　ファイル名をキーボードから入力

ファイル名は、ダイアログボックス内のファイル一覧のボックスのなかから選択することもできます。

マウスカーソルまたはカーソルをファイル一覧のボックスのなかに移動して、読み出すファイルをアクティブポインタで選択します。

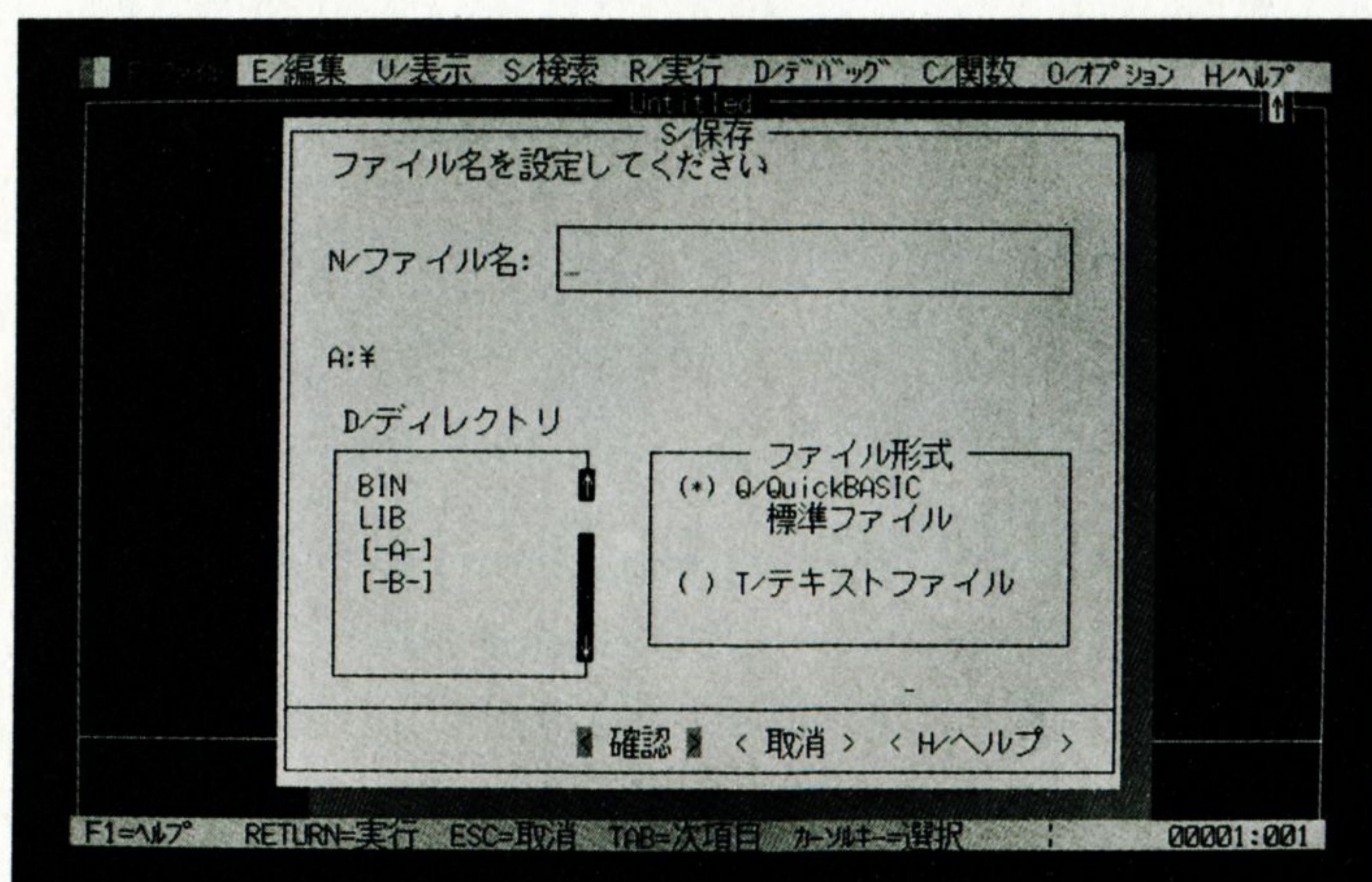
プログラムの追加読込　　**M／結合...**　現在編集中のプログラムに、フロッピーディスクから別のファイルを読み出して挿入します。

このコマンドを選択すると、「O／読込」と同じようにダイアログボックスが開いて、読み出すプログラムのファイル名を聞いてきますから、同様にフロッピーディスクからパソコンに読み出すファイルを指定します。

読み出したファイルは、編集中のプログラムのカーソルがある位置から後ろに挿入されます。

他のプログラムで使われているルーチンなどをつけ加えるためなどに利用します。

プログラムの保存　　**S／保存**　現在編集対象になっているプログラムをフロッピーディスクに保存します。

すでに名まえがつけられたプログラムの場合は、そのままタイトルバーに表示されているファイル名で保存されます。

■ダイアログボックス（保存）

M／結合	A／名前を変えて保存	D／MS-DOS コマンド
S／保存	V／すべて保存	EXIT
N／ファイル名	P／印刷	X／終了

名まえのついていない新しいプログラムを保存する場合は、ダイアログボックスが開いてファイル名を聞いてきます。

N／ファイル名　ファイル名をキーボードから入力

保存するファイル名は、すでにフロッピーディスクにあるファイルの名まえと同じにならないように注意します。同じ名まえを指定すると、以前のファイルがなくなって、いま保存したプログラムに置き替わってしまいます。

拡張子になにも指定しないと、自動的に「.BAS」の拡張子がつけられて保存されます。

プログラムの名まえを変えて保存　**A／名前を変えて保存...**　現在編集対象になっているプログラムの名まえを変更して、フロッピーディスクに保存します。

このコマンドを選択すると、すでに名まえのついているファイルでもダイアログボックスが開いて、ファイル名を聞いてきます。「S／保存」で、新しい名まえをつける場合と同様の手順でファイル名を設定します。

すでにあるプログラムを利用して、新しいプログラムをつくった場合などに利用します。

全モジュールの保存　**V／すべて保存**　すべてのモジュールをフロッピーディスクに保存します。

マルチモジュールプログラムの場合、複数のモジュールが読み込まれています。読み込まれたすべてのモジュールをひとつのプログラムとしてまとめてフロッピーディスクに保存するのに使われます。

プログラムリストの印刷　**P／印刷**　現在編集対象になっているプログラムのリストを印刷します。

このコマンドを選択すると、ダイアログボックスが開いて、プリンタで印刷するプログラムリストの範囲を聞いてきます。

S／指定範囲	あらかじめ指定した範囲を印刷
W／アクティブウィンドウ	現在アクティブになっているウィンドウのリストのみを印刷
M／カレントモジュール	現在編集対象になっているモジュールのみ

を印刷

A／全体　　読み込まれているすべてのプログラムのリストを印刷

MS-DOSのコマンドの実行　**D／MS-DOSコマンド**　Quick BASICを一時中断して、MS-DOSのコマンドを実行できるようにします。

このコマンドを選択すると、Quick BASICを一時的に中断して、MS-DOSのコマンドラインに移ります。

Quick BASICを終了したわけではないので、MS-DOSのコマンドを利用したあとは、MS-DOSのコマンドライン「A>」などの状態から、

EXIT↵

とすると、またQuick BASICの中断前の状態に戻ることができます。

Quick　BASICを利用しているときに、MS-DOSのDIR、DEL、COPY、FORMATなどのコマンドを使う必要がある場合に利用します。

ただし、日本語FEPを利用していて、大きなプログラムなどを作成編集している場合などは、メモリ不足になってこのコマンドが利用できない場合があります。

Quick BASICの終了　**X／終了**　Quick BASICを終了してMS-DOSに戻ります。

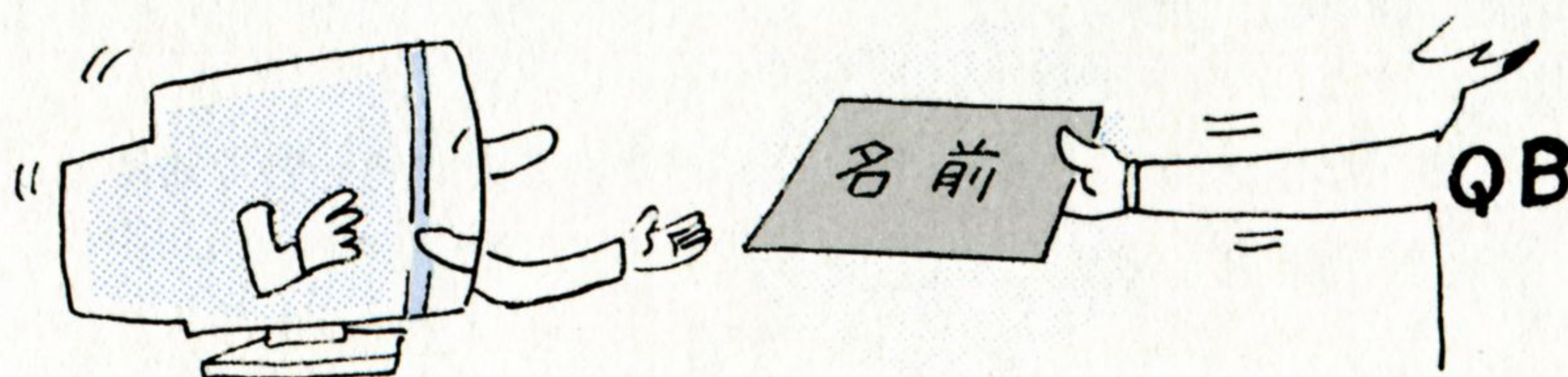
このコマンドを選択すると、読み込まれたプログラムに変更が加えられていない場合は、そのままMS-DOSのコマンド待ちの状態に戻ります。

プログラムに編集などで変更が加えられた場合には、ダイアログボックスが開かれて、ファイルを保存するかどうかの確認がされます。

Y／はい　　プログラムを保存して終了する

N／いいえ　　プログラムを保存せずに終了する

取消　　終了せずにエディタ画面に戻る

いずれかのボックスを選択します。

構造管理する

構造的に管理されるプログラムやモジュール

プログラムの構造

Quick BASICでは、プログラムを「ブロック」単位で構成することを強く意識しています。

こども用の教育玩具にある「○○ブロック」を考えてみるとわかりやすくなります。

プログラムは、小さなブロックを単位として、部品であるさまざまな形をしたたくさんのブロックを組み合わせてつくっていくという考え方です。

Quick BASICのブロックは、パソコンで行う処理の単位で、ひとつの作業の手順をまとめたプログラムの部品です。

小さなブロックから大きなブロックへ、部品と部品を組み合わせて目的のしごとをするプログラムを構成していきます。

モジュールは、こうしていくつかのブロックを組み合わせてできた部品を、さらに集めて組み立てた「モジュールレベルコード」と「プロシージャ」から構成されています。

プロシージャには、その働き方で「SUB」と「FUNCTION」の2種類があります。

ひとつのモジュールだけでできているプログラムを「シングルモジュールプログラム」といい、複数のモジュールでできているプログラムを「マルチモジュールプログラム」といいます。

先頭のモジュールを「メインモジュール」といい、シングルモジュールプログラムは、メインモジュールだけで構成されるプログラムということができます。

■プログラムの構成

プログラム

メインモジュール

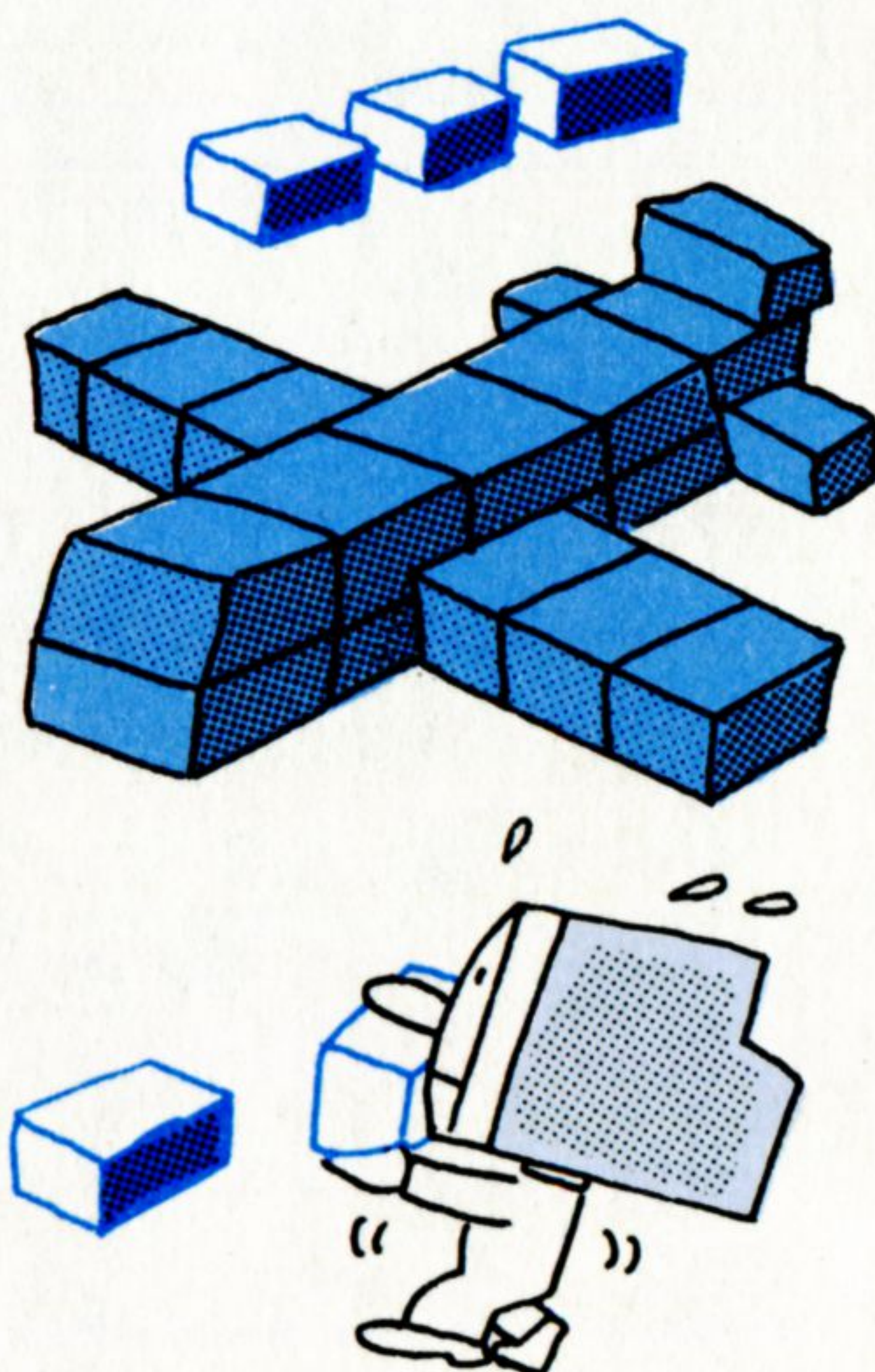
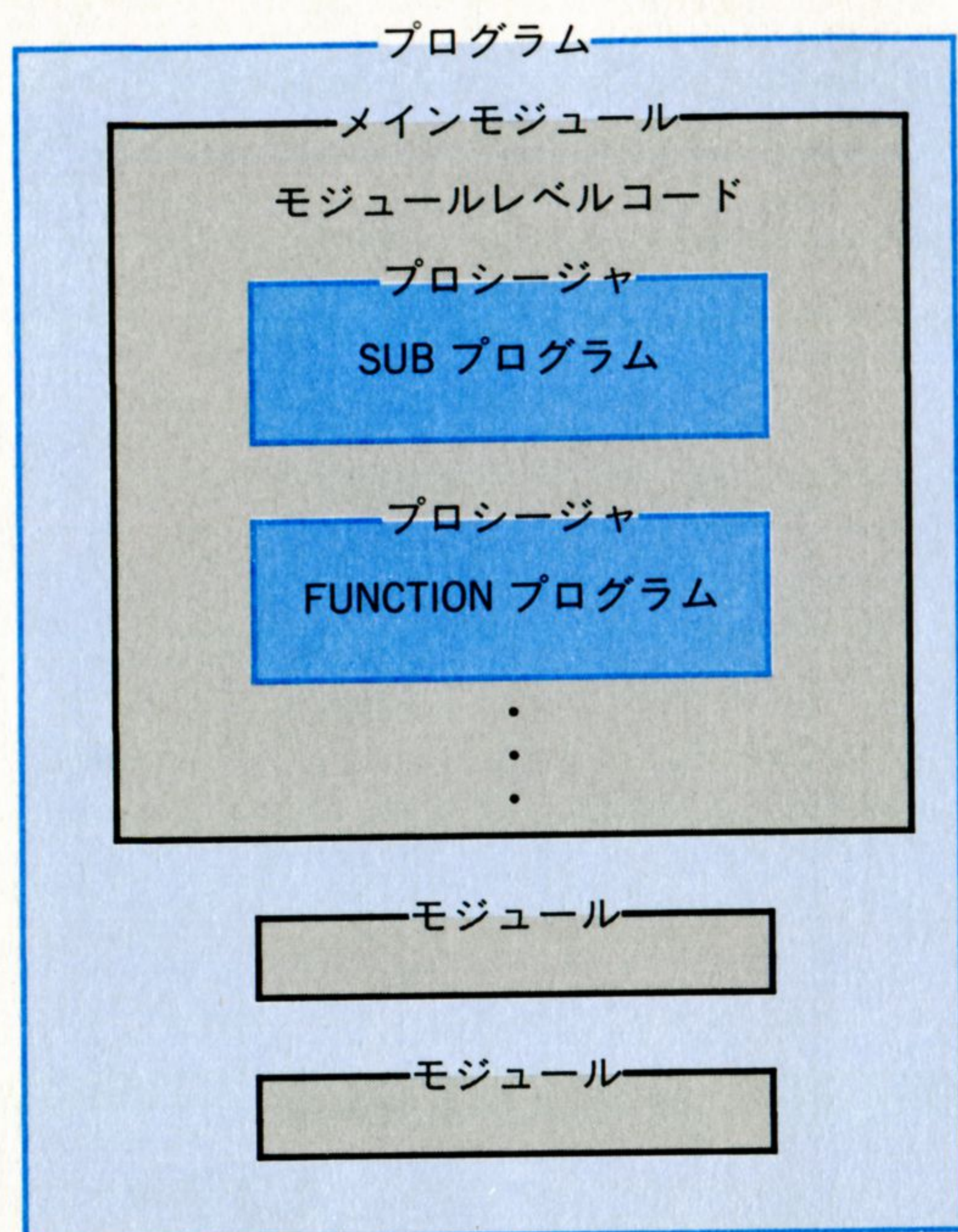
モジュールレベルコード

プロシージャ

SUB プログラム

プロシージャ

FUNCTION プログラム

・
・
・

モジュール

モジュール

表示メニュー

Quick BASIC では、モジュールやプログラムが構造化されて管理されています。すべてのプログラムを画面上に同時に表示することができなくても、見たいときにはいつでも呼び出せるような機能が整っています。

プログラム編集中やテストランの際に必要な表示に関するコマンドが、「V／表示」メニューに配置されています。

プロシージャの一覧　**S/SUB 一覧...** f・2　読み込まれているプロシージャの一覧を表示します。

現在 Quick BASIC のエディタに読み込まれているプログラムのモジュールや、プロシージャの一覧を表示して、プロシージャを操作するためのダイアログボックスを開きます。

読み込まれているモジュールのファイル名は、大文字で表示されます。モジュールのなかにあるプロシージャは、1文字分下げて、ファイル名の下にグループ化されて表示されます。ウィンドウに呼び出すプロシージャを選択して、

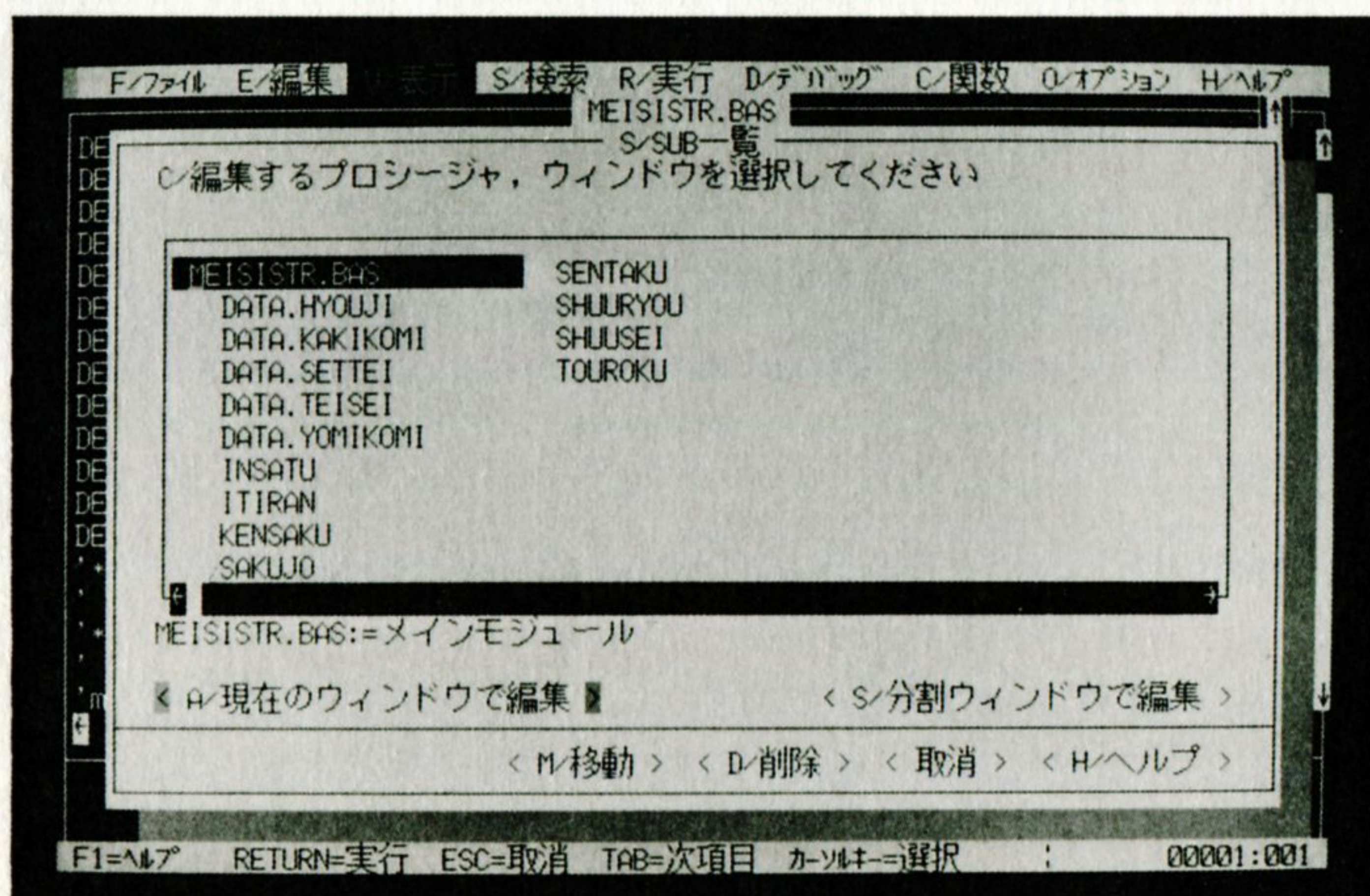
プロシージャ　V／表示
プログラムブロック　マルチモジュール　S／SUB一覧
モジュールレベルコード　シングルモジュール　E／次のSUB

呼び出すウィンドウを指定します。また、プロシージャの操作は、操作するプロシージャをマウスや[TAB]キーでアクティブポインタを移動させて選択し、移動または削除の指定をします。

C／プログラム選択	読み込まれているプログラムの一覧が表示され、このなかからプログラムを選択
W／アクティブウィンドウ	選択したプログラムをアクティブウィンドウに表示
S／分割ウィンドウ	ウィンドウを分割して新たに開いたウィンドウに選択したプログラムを表示
M／移動	選択したプログラムを他のモジュールに移動
D／削除	選択したプログラムをモジュールから削除

次のサブプログラムの表示　**E／次のSUB**　[SHIFT]＋[f・2]　次のサブプログラムをアクティブウィンドウに表示します。

SUBプログラムは、アルファベット順に管理されています。このコマンドを選択すると、現在編集対象になっているプロシージャの次の順番のプロシージャがアクティブウィンドウに呼び出されて表示されます。

```
 F/ファイル  E/編集  V/表示  S/検索  R/実行  D/デバッグ  C/関数  O/オプション  H/ヘルプ
                          MEISISTR.BAS
                            S/SUB一覧
   C/編集するプロシージャ，ウィンドウを選択してください

     MEISISTR.BAS            SENTAKU
       DATA.HYOUJI           SHUURYOU
       DATA.KAKIKOMI         SHUUSEI
       DATA.SETTEI           TOUROKU
       DATA.TEISEI
       DATA.YOMIKOMI
       INSATU
       ITIRAN
       KENSAKU
       SAKUJO

   MEISISTR.BAS:=メインモジュール

   < A/現在のウィンドウで編集 >          < S/分割ウィンドウで編集 >

            < M/移動 >  < D/削除 >  < 取消 >  < H/ヘルプ >
 F1=ヘルプ  RETURN=実行  ESC=取消  TAB=次項目  カーソルキー=選択    :    00001:001
```

■ダイアログボックス（プロシージャ一覧）

いままで編集対象だったプロシージャは、画面からは消えてしまいますが、そのままサブプログラムのひとつとして管理されていますから、なくなってしまうわけではありません。

アクティブウィンドウの分割　**P／画面分割**　アクティブウィンドウを上下２画面に分割します。このコマンドを選択すると、アクティブウィンドウが上下２画面に分割され、編集対象だったプログラムが、両画面に表示されます。

分割された画面には、別のプロシージャを呼び出すこともできます。両画面ともそれぞれに編集することができます。

このコマンドは、切り替えスイッチ方式で、一度選択すると画面が分割され、次に選択されるともとどおりのひとつの画面に戻ります。

編集対象のプログラムのあるアクティブウィンドウを切り替えるには、ショートカットキー

f・6	**窓切り替え（後）**
SHIFT ＋ f・6	**窓切り替え（前）**

で切り替えます。

次に実行されるステートメント　**N／次のステートメント**　プログラムを一時中断したときにこのコマンドを実行すると、カーソルが次に実行されるステートメントのある行に移動します。プログラムは実行されるわけではなく、実行中断された位置にカーソルを移動するだけですので、プログラムのよその場所からカーソルを戻すことにも利用することができます。

```
 F/ファイル  E/編集  V/表示  S/検索  R/実行  D/デバッグ  C/関数  O/オプション  H/ヘルプ
                              MEISISTR.BAS
DECLARE FUNCTION SENTAKU! ()
DECLARE SUB TOUROKU ()
DECLARE SUB SHUUSEI ()
DECLARE SUB SAKUJO ()
DECLARE SUB KENSAKU ()
DECLARE SUB ITIRAN ()
DECLARE SUB INSATU ()
DECLARE SUB SHUURYOU ()
                              MEISISTR.BAS
            CASE ELSE
                COLOR 6
                LOCATE MenuKosuu * 2 + 4, 21
                PRINT "1～7の番号で選択してください"
                COLOR 7
                Kari$ = INPUT$(1)
        END SELECT
    LOOP WHILE Owari = Iie
    CLS
                              ダイレクト モード
<SHIFT+F1=ヘルプ> <F6=窓切換> <F2=SUB一覧> <F5=実行> <F8=トレース> ¦      00008:033
```

■画面分割

U／実行画面　E／編集
P／画面分割　I／インクルードファイル編集　S／新規SUB
N／次のステートメント　L／インクルードファイル表示　S／新規FUNCTION

実行画面を表示　**U／実行画面**　プログラムを実行して、エディタ画面に戻っているときにこのコマンドを実行すると、プログラムの実行画面を表示することができます。実行画面を見てから、なにかのキーを押すか、マウスを左クリックするとエディタ画面に戻ることができます。

インクルードファイルの編集　**I／インクルードファイル編集**　アクティブウィンドウにインクルードファイルを表示して、編集対象にします。

インクルードファイルとは、マルチモジュールプログラムを作成する場合に、複数のモジュールで共通に利用するサブプログラムや関数の型などの宣言ステートメントをまとめておくファイルです。

インクルードファイルの表示　**L／インクルードファイル表示**　アクティブウィンドウにインクルードファイルを表示します。

このコマンドは、切り替えスイッチ方式になっていて、もう一度選択するとインクルードファイルの表示をやめて、もとの画面に戻ることができます。

サブプログラムの編集

SUB・FUNCTIONのサブプログラムは、メインプログラムの作成中でも、

SUB⏎ または、FUNCTION⏎

として、自動的に作成を開始することができますが、新しいサブプログラムを作成するコマンドは、「E／編集」のメニューにも配置されています。

●編集メニュー　E／編集

新規SUBプロシージャの作成　**S／新規SUB...**　新しいSUBプロシージャを作成します。

新しいSUBプロシージャを作成するときにこのコマンドを選択します。コマンドを選択するとダイアログボックスが開いて、SUBプロシージャの名まえを聞いてきますから、入力します。

N／ファイル名　SUBプロシージャの名まえを入力

新規FUNCTIONプロシージャの作成　**S／新規FUNCTION...**　新しいFUNCTIONプロシージャを作成します。

新しくFUNCTIONプロシージャを作成するときにこのコマンドを選択し

ます。コマンドを選択するとダイアログボックスが開いて、FUNCTION プロシージャの名まえを聞いてきますから、入力します。

N／ファイル名　FUNCTION プロシージャの名まえを入力

複数モジュールのファイル管理

複数モジュールプログラムの読み出しや削除といったファイル管理は、「F／ファイル」のメニューで行います。

●ファイルメニュー　　F／ファイル

サブファイルの作成　　**C／サブファイル作成...**　サブモジュール、インクルードファイル、ドキュメントファイルを作成します。

マルチモジュールプログラムで、メインモジュール以外のモジュールやインクルードファイル、ドキュメントファイルを作成する場合に選択します。

このコマンドを選択すると、ダイアログボックスが開いて、作成するモジュールなどの名まえ、ファイルの形式を聞いてきます。

N／ファイル名　　モジュールなどのファイル名を入力

形式の選択

M／モジュール　　モジュール作成の場合、＊をつける

I／インクルード　インクルードファイル作成の場合、＊をつける

D／ドキュメント　ドキュメントファイル作成の場合、＊をつける

サブファイルの読み出し　　**L／サブファイル読込...**　サブモジュール、インクルードファイル、ドキュメントファイルをフロッピーディスクから読み出します。

マルチモジュールプログラムで、メインモジュール以外のモジュールやインクルードファイル、ドキュメントファイルをフロッピーディスクから読み出します。このコマンドを選択すると、ダイアログボックスが開いて、読み出すモジュールなどの名まえ、ファイルの形式を聞いてきます。

N／ファイル名　　モジュールなどのファイル名を一覧表から選択

形式の選択

M／モジュール　　モジュールとして読み出す場合、＊をつける

I／インクルード　インクルードファイルとして読み出す場合、＊をつける

D／ドキュメント　ドキュメントファイルとして読み出す場合、＊をつ

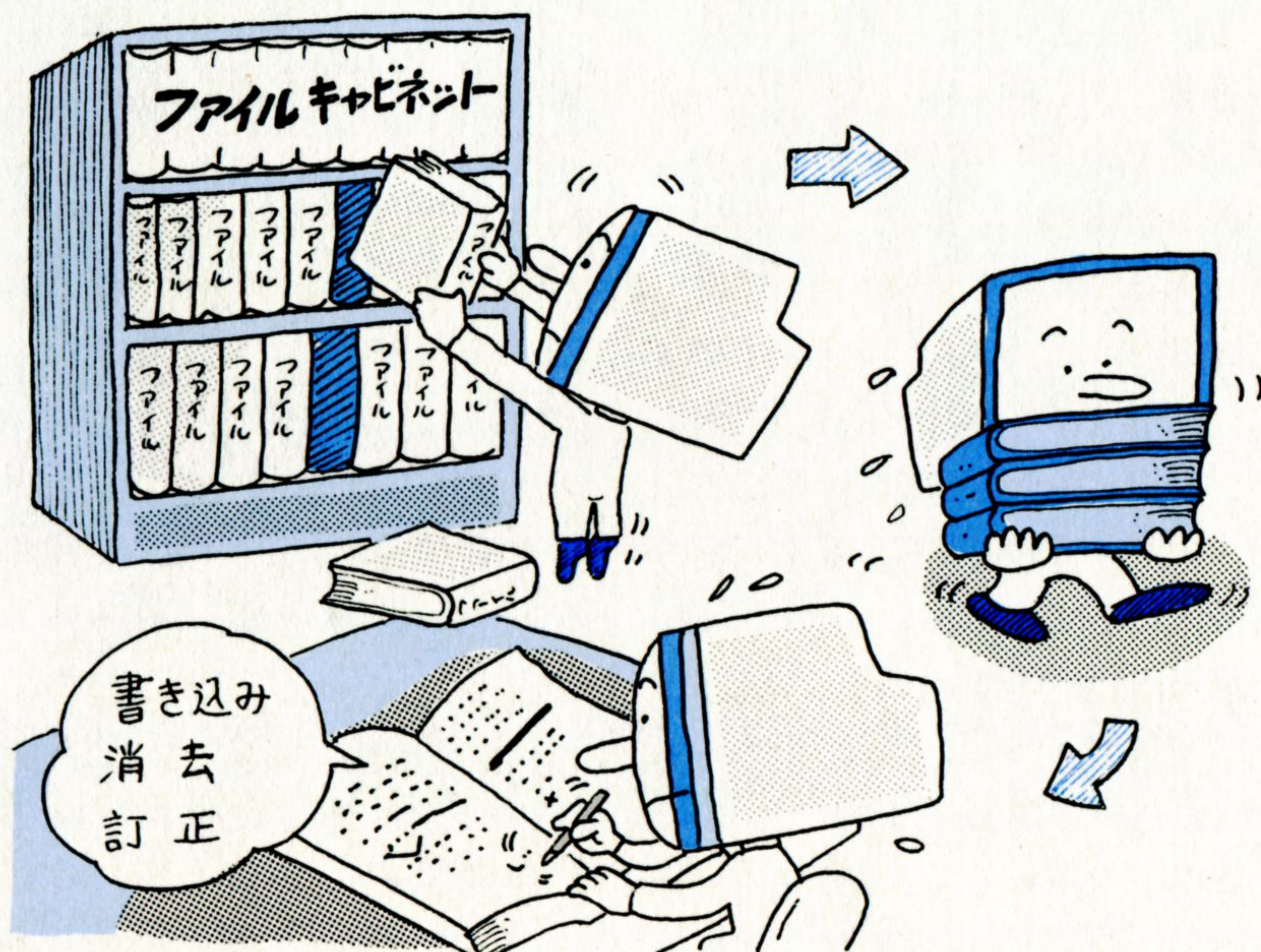
ける

サブファイルの消去　**U／サブファイル解放...**　読み出されたサブモジュール、インクルードファイル、ドキュメントファイルを消去します。

すでに読み出されているマルチモジュールプログラムで、メインモジュール以外のモジュールやインクルードファイル、ドキュメントファイルを消去します。

このコマンドを選択すると、ダイアログボックスが開いて、消去するモジュールなどの名まえを聞いてきますから、一覧表から選択します。

メインプログラムを削除した場合は、どのモジュールをメインプログラムにするか、さらにダイアログボックスが開いて聞いてきますから、一覧表から選択します。

ファイルはメモリからは消去されても、フロッピーディスクから削除されるわけではありません。フロッピーディスクには、読み出したときのファイルが残っています。

デバッグする

プログラムの害虫を駆除するデバッグコマンド

プログラムを作成していると、たとえ小さなプログラムでも、まちがいやかん違い、考え違いをしていて、最初から正しいプログラムをつくるのはなかなか難しいものです。

プログラムも人間の行うことですから、当然ミスがあります。ところが、プログラムにミスがあると、石頭のパソコンは動いてくれません。

しかし、パソコンが動いてくれないミス、つまりQuick BASICが発見してくれるミスは、すぐに訂正できるのでまだよいほうです。

BASICの文法にはかなっていても、考え違いをしていて、意図したことと違う動作をするようなミスがあった場合は、発見が非常に難しいものです。

プログラムの単純なミスから、このような発見の難しいミスまでを含めて、プログラムの誤りを取り除き、正常に動作するプログラムに完成させるのがデバッグです。

デバッグは、英語で「DEBUG」と書きます。BUGは「虫」のこと、DEは「否定の接頭辞」ですから、「虫をとる」ぐらいの意味になり、日本語でも「虫とり」などという場合もあります。

Quick BASICでは、構文チェック機能が働いていて、実行不可能なステートメントが入力されると、ダイアログボックスを開いて「構文エラー」を知らせてくれます。

また、実行時にもエラーを発見すると、実行をストップさせて、エラーのあったステートメントを示してくれます。

バグを発見するにはなによりQuick BASICのデバッグコマンドを活用するのがよいでしょう。

デバッグメニュー

Quick BASICには、プログラムに潜むバグを発見するために、非常に強力なデバッグ機能が用意されています。この強力な機能を「デバッガ」と呼びます。

Quick BASIC以外の言語では、このようなデバッグを行うために、多くの場

合専門の「デバッガ」というプログラムが用意されています。

普通は、このデバッガにプログラムを通してバグを発見し、再度エディタを起動して修正して、またデバッガにかけるということを何度も繰り返しています。

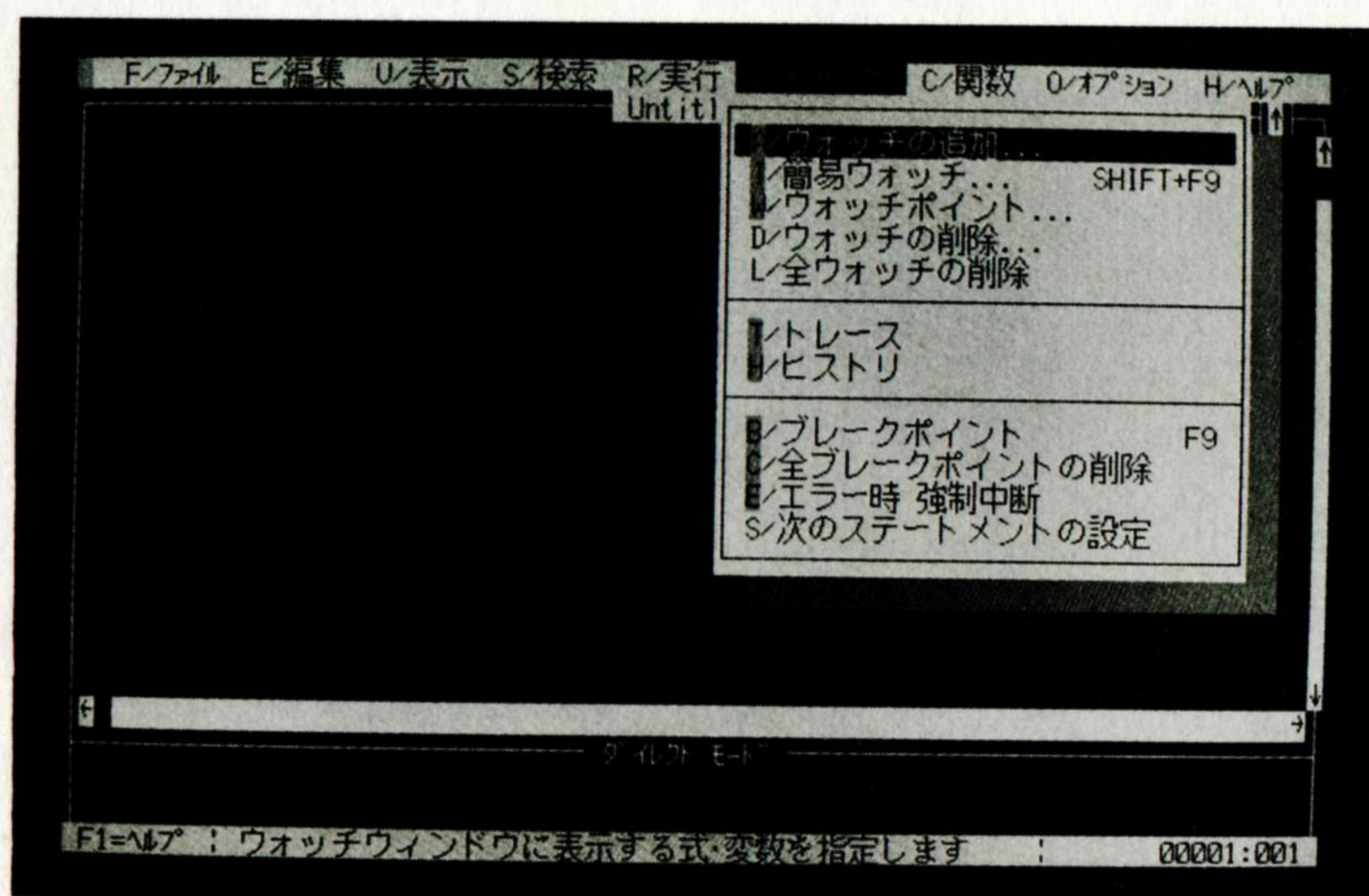
しかし、Quick BASICでは、このデバッガの機能も統合環境のなかに取り入れて、プログラムをつくりながら、動かしながらデバッグもできるようになっています。Quick BASICでプログラムを実行中にデバッガの機能を使う方法は簡単です。

STOPキーを押すと、プログラムの実行を一時停止し、エディタ画面であるビューウィンドウに画面が切り替わります。

画面が切り替わらない場合は、つづけてスペースバーなどなにか適当なキーを押すと、ビューウィンドウに画面が切り替わります。

エディタ画面に切り替わると、それまで実行していた部分のプログラムリストが表示されますから、すぐにデバッグ作業に入ることができます。

デバッグ機能は、「D／デバッグ」のメニューに配置されています。

ウォッチ式の設定　**A／ウォッチの追加...**　値の変化を監視する変数を設定します。

■プルダウンメニュー（デバッグ）

このコマンドを選択すると、ダイアログボックスが開いて、ウォッチウィンドウに設定する式の入力を求めてきます。

ウォッチウィンドウに設定する式の入力　　監視する変数名を入力

ウォッチ式とは、変数の監視機構のことです。プログラム実行中に変数の値の変化するようすを監視しています。

このコマンドでウォッチ式を設定すると、ビューウィンドウの上に「ウォッチウィンドウ」が開いて、設定した式と値が表示されます。

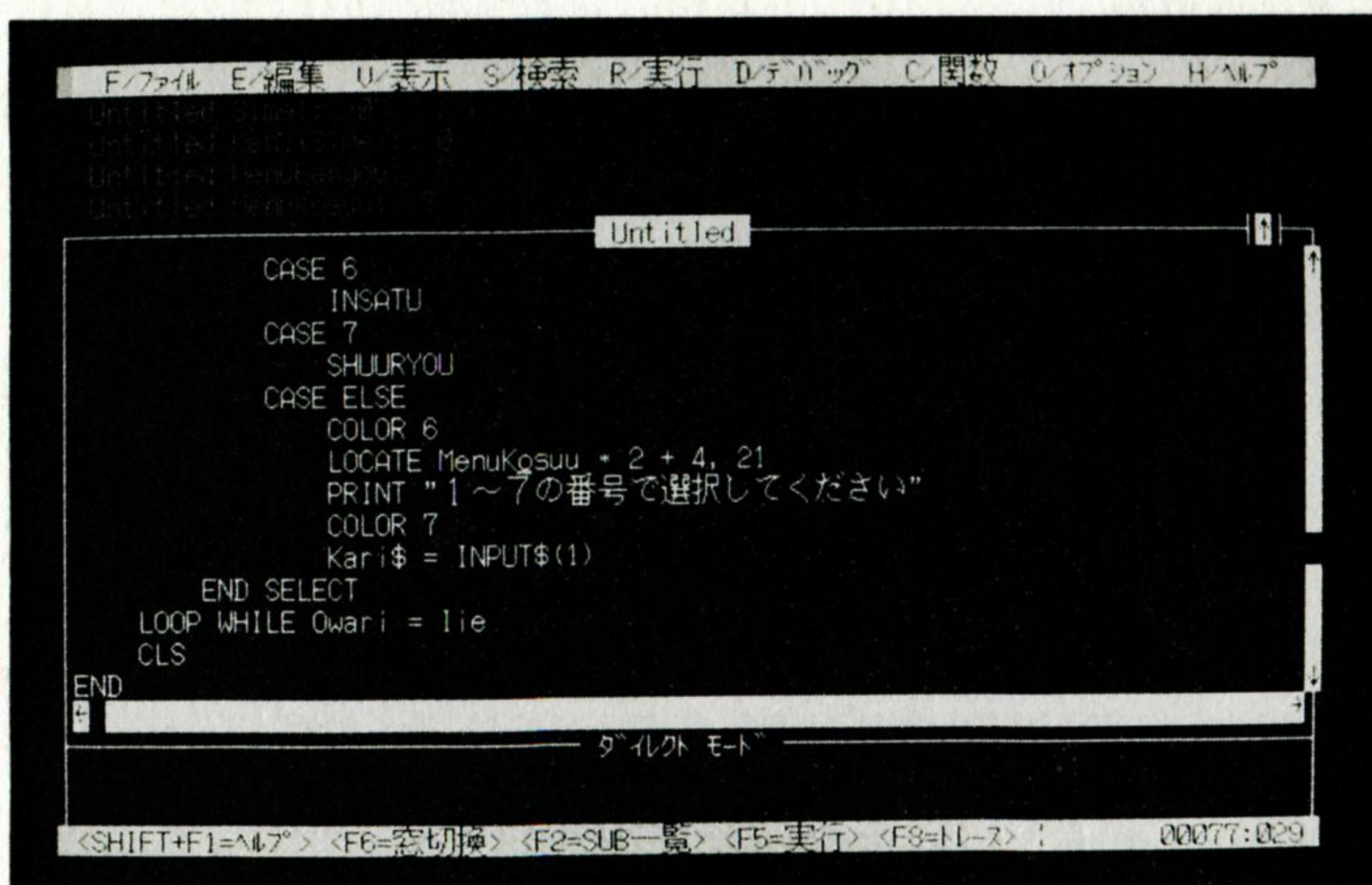
設定したプロシージャ以外のプロシージャを実行しているときには、式の値として「Not watchable」と表示されて、監視できないことを示します。

簡易ウォッチの設定　　**I／簡易ウォッチ...**　　プログラム中の変数にカーソルを合わせてこのコマンドを実行すると、ダイアログボックスが現れて、その変数と値が表示されます。また、式を指定すると式と式の値が表示されます。SHIFT＋f・9 のショートカットキーにこの機能が設定されています。

ダイアログボックスのなかから「A／ウォッチの追加」を選択すると、指定した変数や式を、ウォッチウィンドウに追加して設定することができます。

ウォッチポイント　　**W／ウォッチポイント...**　　ウォッチポイントの式を設定します。このコマンドを選択すると、ダイアログボックスが開いて、ウォッチポイントに設定する式の入力を求めてきます。

条件式の設定：（式＝真になるまで実行します）　ウォッチポイントに設定する式の入力

```
 F/ファイル  E/編集  V/表示  S/検索  R/実行  D/デバッグ  C/関数  O/オプション  H/ヘルプ
                                Untitled
            CASE 6
                INSATU
            CASE 7
                SHUURYOU
            CASE ELSE
                COLOR 6
                LOCATE MenuKosuu * 2 + 4, 21
                PRINT "1～7の番号で選択してください"
                COLOR 7
                Kari$ = INPUT$(1)
        END SELECT
    LOOP WHILE Owari = lie
    CLS
END
                              ダイレクト モード
<SHIFT+F1=ヘルプ> <F6=窓切換> <F2=SUB一覧> <F5=実行> <F8=トレース> ¦      00077:029
```

■ウォッチウィンドウ

A／ウォッチの追加　ウォッチ式　L／全ウォッチの削除
I／簡易ウォッチ　ウォッチウィンドウ　T／トレース
D／デバッグ　W／ウォッチポイント　D／ウォッチの削除　H／ヒトスリ

ウォッチポイントは、設定した式の値が「真」つまり「0でない値」になったときに、プログラムの実行を停止します。

ウォッチウィンドウには、設定した式が表示されます。「FALES」(偽)または「TRUE」(真)が表示されて、現在の状態がわかります。

設定したプロシージャ以外のプロシージャを実行しているときには、「Not watchable」と表示されて、監視できないことを示します。

ウォッチの削除　**D／ウォッチの削除...**　ウォッチウィンドウに設定されているウォッチ式を削除します。

監視する必要のなくなったウォッチ式を削除します。このコマンドを選択すると、ダイアログボックスが開いて、設定されているウォッチ式の一覧が表示されます。

一覧のなかから選択したウォッチ式が削除されます。

全ウォッチの削除　**L／全ウォッチの削除**　ウォッチウィンドウに設定されているすべてのウォッチ式を削除します。

このコマンドを選択すると、確認を求めずに設定されているすべてのウォッチ式が削除されますから注意しなければなりません。

トレース　**T／トレース**　トレースモードを切り替えます。

トレースは、プログラムの実行状態を1ステートメントずつゆっくりと追いかけて、実行しているステートメントを強調表示します。

このコマンドは、切り替え式のスイッチになっていて、初めにこのコマンドを選択すると、トレースオンの状態になり、コマンドの前に「>」のチェックマークがつきます。

次にこのコマンドを選択すると、トレースモードがキャンセルされて、トレースオフの状態になり、チェックマークも消えます。

ヒストリ　**H／ヒストリ**　ヒストリモードを切り替えます。

ヒストリは、ステートメントの実行順序を逆順、正順に追っていくための機能です。

この機能を選択すると、Quick BASICは最後に実行した20個のステートメントを記憶するようになります。

プログラムを一時停止させて、次のようなファンクションキーで操作します。

SHIFT ＋ f・8	ヒストリ（前）	実行ステートメントを逆順にさかのぼる
SHIFT ＋ f・10	ヒストリ（後）	実行ステートメントを正順にたどる

トレースと違って、実際にステートメントを実行するわけではありませんが、ファンクションキーを押すたびに、実行されたステートメントにカーソルが移りますから、プログラムがどのように実行されたかを知ることができます。このコマンドも切り替え式スイッチ方式になっています。

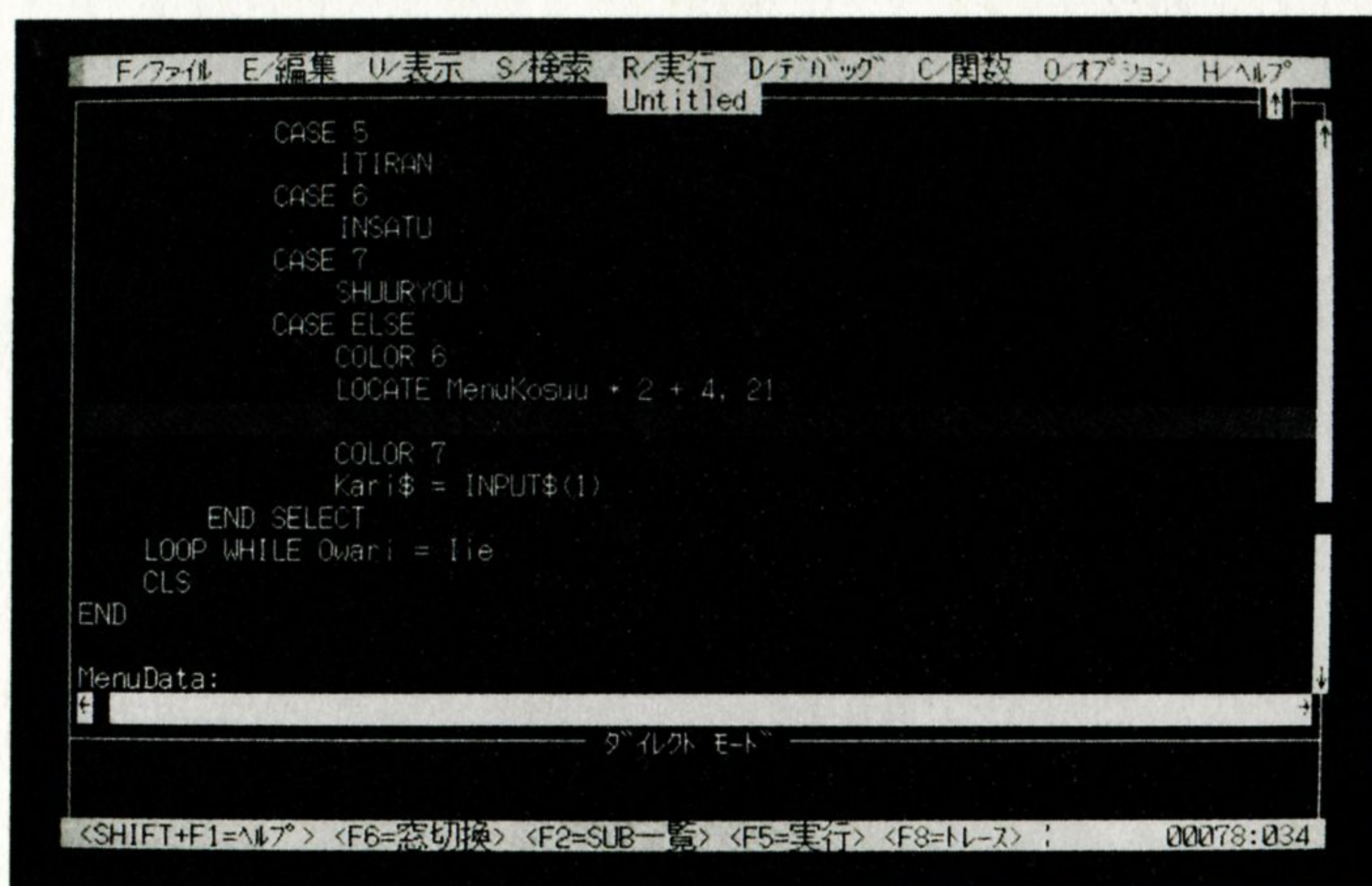
ブレークポイントの設定　**B／ブレークポイント**　f・9　プログラムの一時停止の位置を設定します。

ブレークポイントを設定すると、このポイントで指定されたステートメントで、プログラムの実行を一時停止して、画面をビューウィンドウに切り替えます。

プログラムを一時停止して、ダイレクトモードを利用して変数の内容などを確認したり、変更してプログラムの誤りを見つけ出すのに利用します。

プログラムを一時停止したい行にカーソルを合わせて、このコマンドを選択すると、設定した行が強調表示されます。

このコマンドも切り替え式スイッチになっていて、ブレークポイントが設定されている行にカーソルを合わせて、再度コマンドを選択すると、ブレークポ

```
 F/ファイル E/編集 V/表示 S/検索 R/実行 D/デバッグ C/関数 O/オプション H/ヘルプ
                                Untitled
            CASE 5
                ITIRAN
            CASE 6
                INSATU
            CASE 7
                SHUURYOU
            CASE ELSE
                COLOR 6
                LOCATE MenuKosuu * 2 + 4, 21

                COLOR 7
                Kari$ = INPUT$(1)
        END SELECT
    LOOP WHILE Owari = Iie
    CLS
END

MenuData:
                              ダイレクト モード
<SHIFT+F1=ヘルプ> <F6=窓切換> <F2=SUB一覧> <F5=実行> <F8=トレース> ;      00078:034
```

■ブレークポイント

イントの設定を解除します。

全ブレークポイントの解除　**C／全ブレークポイントの削除**　設定されたすべてのブレークポイントを解除します。

このコマンドを選択すると、確認なしに設定されたブレークポイントをすべて解除します。

エラー処理での実行中断　**E／エラー時強制中断**　「ON ERROR」ステートメントを使って、エラートラップを使ったエラー処理をしているときには、エラーが発生するとプログラムの実行がエラー処理ルーチンに移されて、エラー回復の処理を行います。

デバッグ中には、エラー処理を行わずに、プログラムを中断してエラーの原因を突きとめたほうがよい場合があります。このようなときにこのコマンドを指定すると、エラー処理ルーチンを実行せずに、プログラムを強制的に中断してしまいます。ヒストリコマンドと併用して、エラーの発生したステートメントを探し出すのに役立ちます。

続行ステートメントの指定　**S／次のステートメントの設定**　次に実行されるステートメントを設定します。

カレントステートメントは、Quick BASICが次に実行するステートメントです。一度プログラムの実行を中断して、ビューウィンドウに戻ると、強調表示されて示されています。

続行のコマンドが選択されると、このステートメントからプログラムの実行が再開されます。

カーソルを任意のステートメントに合わせて、このコマンドを選択すると、ステートメントが強調表示され、カレントステートメントが移ったことを示します。

次に続行のコマンドを選択すると、このステートメントから実行が再開されます。

プログラムが一時停止した位置から、このコマンドで設定されたカレントステートメントまでを実行するわけではなく、単純に飛ばしてしまうのがこのコマンドの特徴です。

関数メニュー

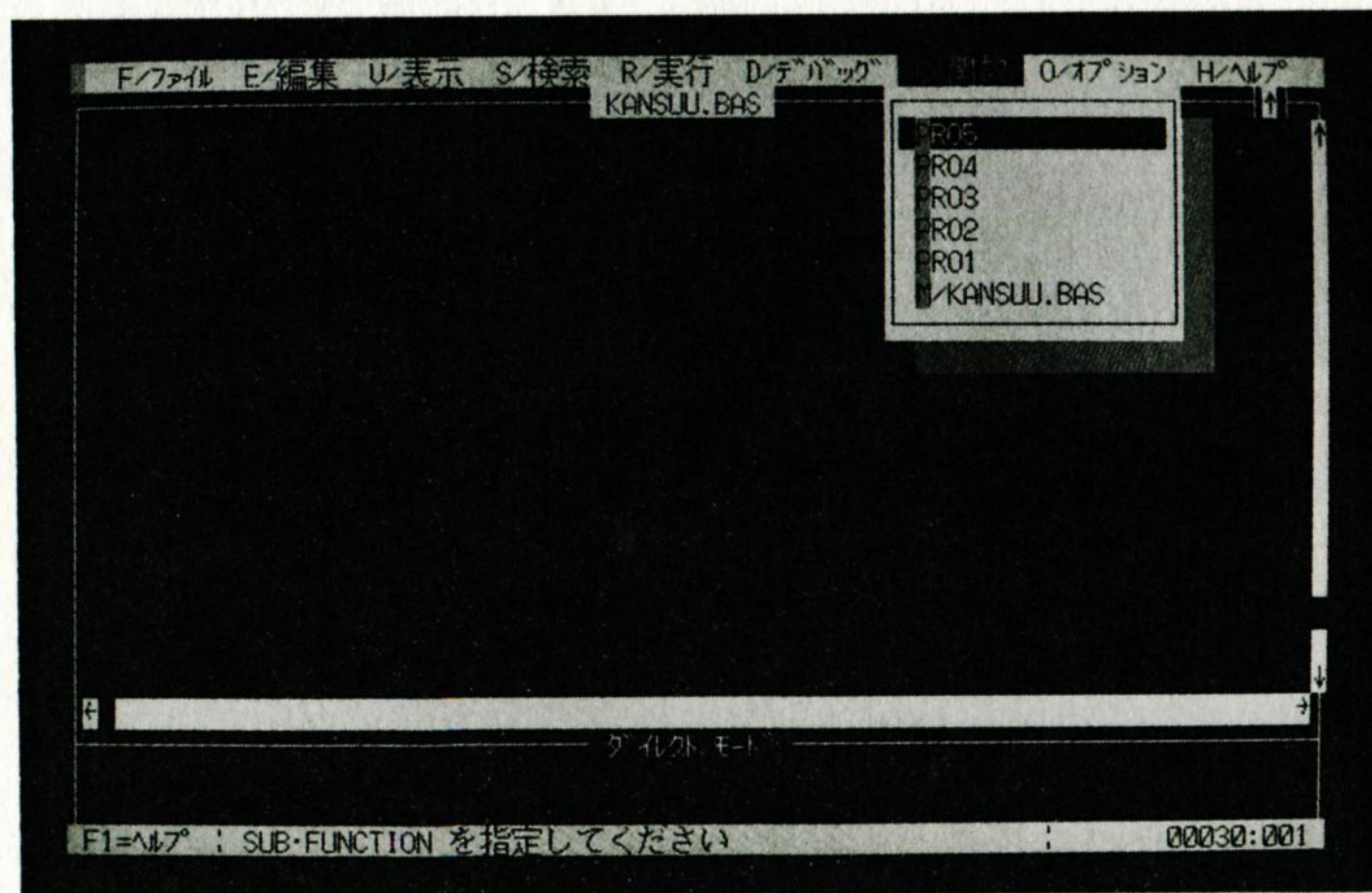
デバッグに役立つメニューに「C／関数」メニューがあります。このメニューは他のメニューと違って、コマンドを選択するためにあるのではありません。

このメニューを選択すると、現在実行中のプロシージャが表示されます。

メインプログラムを実行中は、メインモジュールのファイル名が表示されています。プロシージャが呼び出されているときは、呼び出しの順にリストの下からスタック※状に積み上げられて表示されますから、各プロシージャの呼び出しの関係が下から上へと把握できます。

プロシージャの呼び出しの関係の表示　**C／関数**　プロシージャの呼び出しの関係をスタック状に表示します。

現在実行中のプロシージャがいちばん上に、それを呼び出したプロシージャがその下に、というぐあいに実行プロシージャのリストが表示されます。

アクティブポインタで任意のプロシージャを選択して、⏎キーで確定し、f・7キー（現在行まで実行）を押すと、いちばん上のプロシージャから選択されたプロシージャまでを実行することができます。

■プルダウンメニュー（関数）

※スタック　原義は干し草を積みあげた山のことで、下から積み上げて、上から取り出していくようなことをいいます。
　コンピュータ関係の用語としては、メモリなどに順番に値を積み上げて、積み上げた最後のものから取り出していく（後入れ先出し）方法、またはメモリの領域のことをいいます。

実行する

速度は遅いがデバッグ、修正、追加も自由

プログラムを作成中は、何回もテストランを重ねて、意図したとおりの動作をするかどうかを確かめながら、プログラミングを進めていきます。

Quick BASICでは、作成途中のプログラムを統合環境のなかのQuick BASICインタプリタで動作させます。

ステートメントをひとつずつ解釈実行していきますが、実行速度は、一括してコンパイルした自動実行型プログラムよりは遅いものの、プログラムのデバッグや修正、追加が自由にできます。

こうして、テストをしながらデバッグを繰り返してミスをなくし、Quick BASICコンパイラで完成したプログラムをコンパイルすると、自動実行型のプログラムが完成します。

自動実行型のプログラムは、MS-DOSのコマンドラインから直接実行できますから、アプリケーションソフトやユーティリティソフトと同じ感覚で自作のプログラムを使うことができます。

プログラムの実行やコンパイルに関するコマンドは、「R／実行」メニューに配置されています。

実行メニュー

プログラムの実行、一時停止したプログラムの続行、作成したプログラムのコンパイルなどの機能があります。

プログラムの実行に関するコマンドは、「R／実行」のメニューに配置されています。

プログラムの実行　**S／スタート**　SHIFT＋f・5　現在編集対象となっているプログラムを実行します。

フロッピーディスクからQuick BASICに読み込まれているか、新たに作成中のプログラムを初めから実行します。

変数をクリアして先頭に戻る　**R／リスタート**　一時中断したプログラムの変数などをクリアして、最初のステートメントにカレントステートメントを移

します。

ブレークポイントを設定して、プログラムを一時停止させた場合、すべての変数を0または空の文字列にクリアして、最初に実行可能なステートメントをカレントステートメントに設定して強調表示します。

このあと、任意の位置にカレントステートメントを移して、プログラムを続行することができます。

プログラムの続行　**N／継続**　f・5　一時中断したプログラムを継続して実行します。

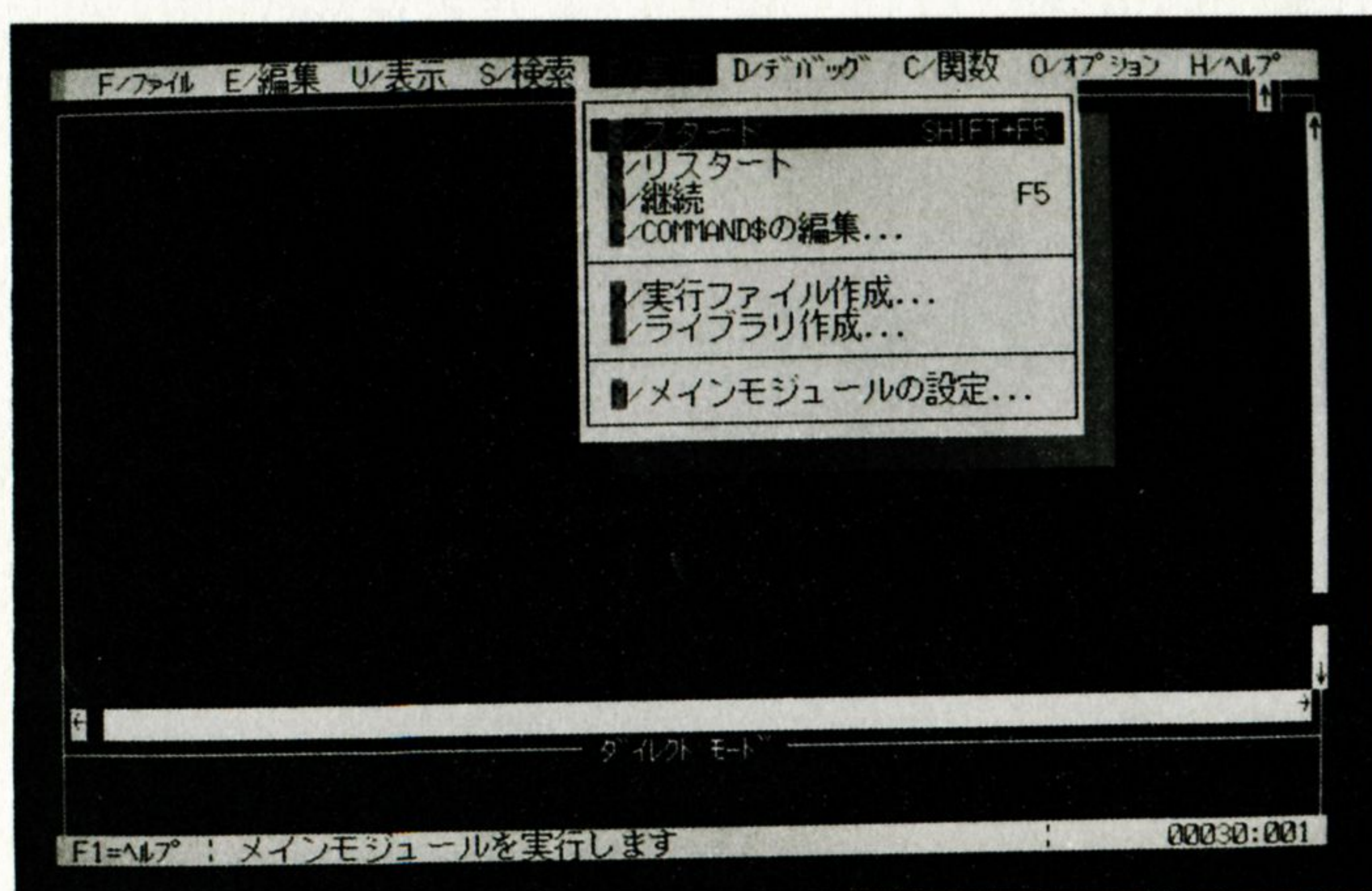
一時中断したプログラムを、中断したステートメントの次のステートメントから継続して実行します。

プログラムが中断されているのでなければ、プログラムの先頭から実行します。

プログラムの実行の制御

プログラムの実行に関するコマンドは、実行コマンドに配置されているものだけではありません。

プログラムの実行制御とデバッグは深く結びついていますから、ファンクションキーや他のメニューの関連するコマンドや操作を概観してみましょう。

指定行までの実行　**カーソル行まで実行**　f・7　現在のステートメントから、プログラム実行上の後方のステートメントにカーソルを合わせてf・7キー

■プルダウンメニュー（実行）

実行する

R／リスタート
R／実行　　　　　　　N／継続
S／スタート　　S／次のステートメントの設定

を押すと、現在のステーメントから指定されたステートメントまでを実行します。

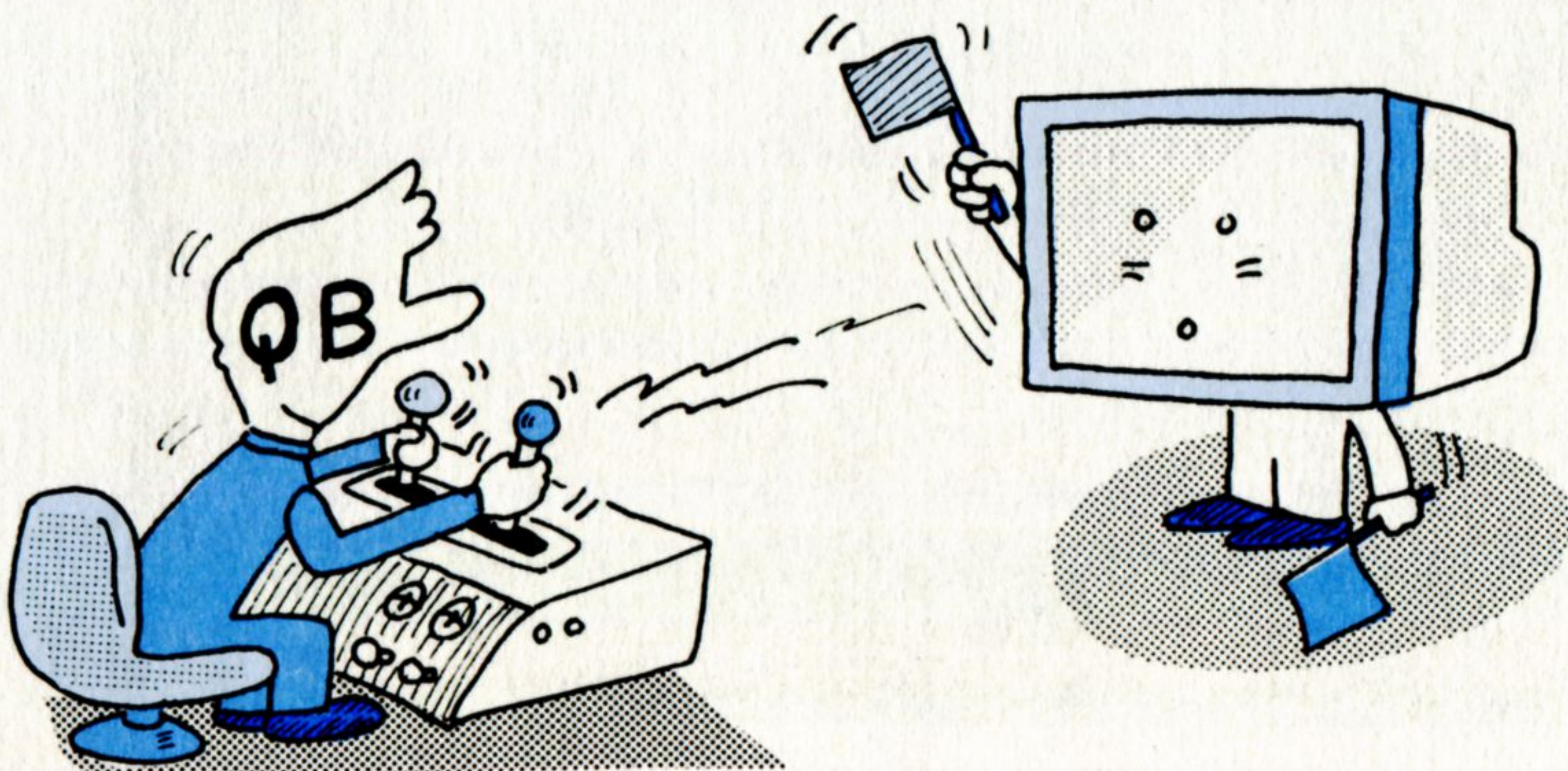
指定行までのスキップ　**S／次のステートメントの設定**　D／デバッグのメニューに配置されています。

シングルステップごとの実行　f・8　プログラムを1ステートメントずつ実行します。

このキーを押すごとに、1ステートメントを実行し、ステートメントが画面表示を行う場合は、短時間だけ実行画面を表示して、またビューウィンドウに戻ります。

プロシージャステップごとの実行　f・10　プログラムの実行をプロシージャを含めて1ステートメントずつ実行します。

このキーを押すごとに1ステートメントを実行し、ステートメントが画面表示を行う場合は、短時間だけ実行画面を表示して、またビューウィンドウに戻ります。

プロシージャを呼び出す場合は、そのプロシージャを1ステートメントとして、プロシージャ内のすべてのステートメントを一度に実行します。

コンパイルする

プログラムを一括してマシン語ファイルに

Quick BASIC の統合環境の下でプログラムの作成、テストラン、デバッグを行ってバグがなくなったら、プログラムをコンパイルして、独立実行型のプログラムにしてみましょう。

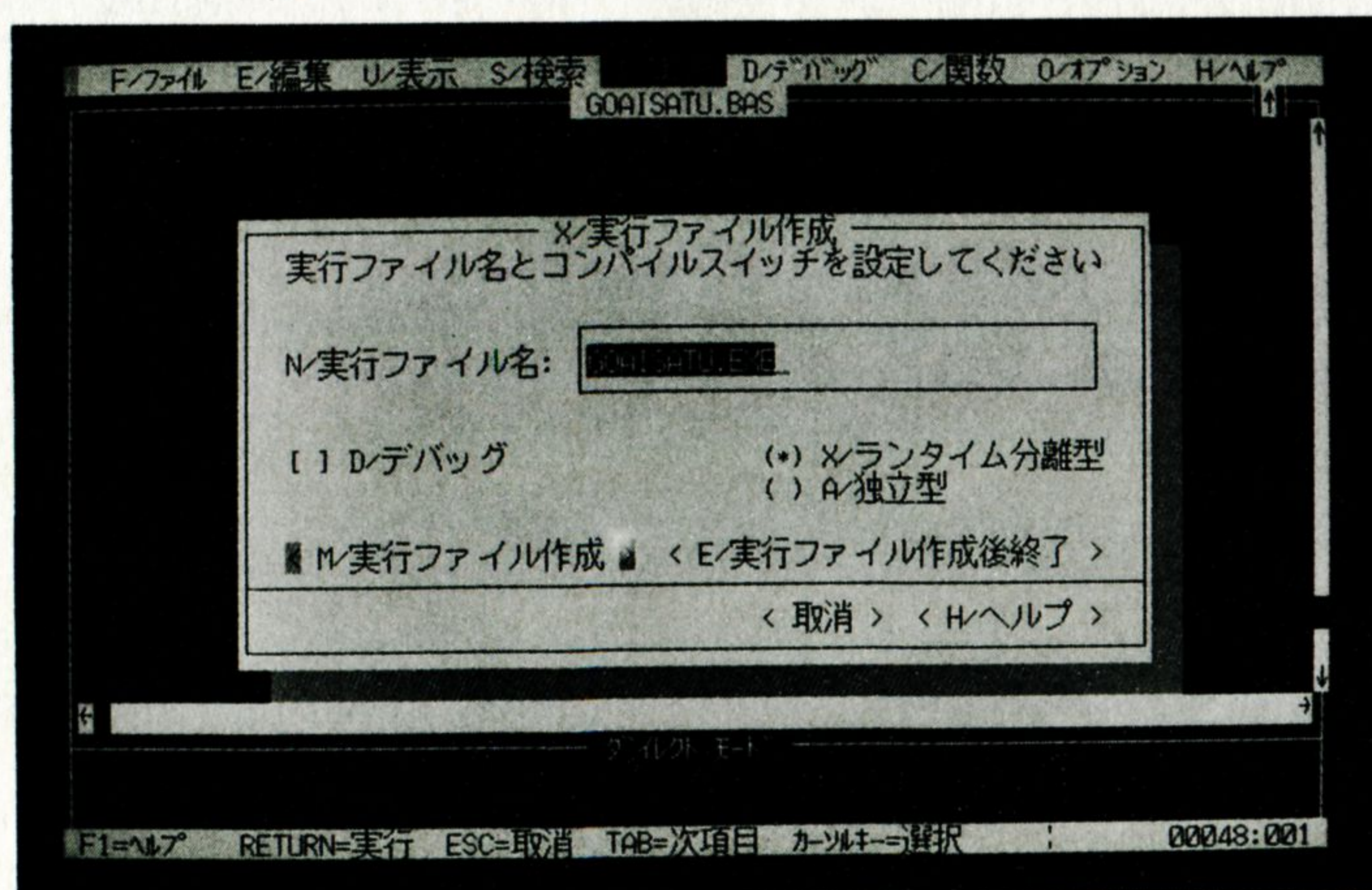
Quick BASIC では、2つの種類の実行型のプログラムを作成することができます。

ひとつは、Quick BASIC のランタイムモジュールを必要とするランタイム分離型 EXE ファイル、もうひとつは、ランタイムモジュールをプログラムのなかに組み込んでしまう独立型 EXE ファイルです。

プログラムのコンパイル

コンパイルは、Quick BASIC のプログラムを、一括してパソコンが直接実行できるマシン語のファイルにしてしまうため、作成したプログラムを、MS-DOS のコマンドラインから実行できるようになります。

コンパイルのコマンドは、「R／実行」のメニューに配置されています。

プログラムのコンパイル　**X／実行ファイルの作成**　現在編集対象になってい

■ダイアログボックス（コンパイル）

るプログラムをコンパイルします。

このコマンドを選択すると、ダイアログボックスが開いて作成するファイルの名まえ、作成形式などを聞いてきます。

●ファイル名

N／実行ファイル名　ファイル名には「.EXE」の拡張子をつける。この拡張子のあるファイルは、MS-DOS が直接実行できるファイルとされる

●ファイルの作成形式

X／ランタイム分離型

ランタイムモジュールの必要なコードサイズの小さい実行可能ファイル

A／独立型　ランタイムモジュールの必要のない実行可能ファイル

●ランタイムエラーのチェック

D／デバッグ　プログラムの実行時にエラーが起きたり、STOPキーが押されたときに実行を停止してエラーコードを返す指定を行う。この指定を行わないでランタイムエラーが発生すると、プログラムの実行は予測不能になる

●コンパイルの開始

M／実行ファイル作成　実行可能ファイルを作成し、Quick BASIC に戻る

E／実行ファイル作成後終了

実行ファイルを作成後、Quick　BASIC を終了して MS-DOS のコマンドラインに戻る

●ランタイム分離型実行ファイル

ランタイム分離型実行ファイルは、プログラムの実行時に「BRUN45A. EXE」と「BRUN45E. EXE」という2つのランタイムモジュールが必要です。

ランタイムモジュールは、コンパイルしたプログラムを実行する際に必要な細かい作業を行う小さなマシン語のルーチンを集めたものです。

このランタイムモジュールとランタイム分離型EXEファイルは、フロッピーディスクのなかで同じディレクトリにあるか、パスを設定したディレクトリになければなりません。

ランタイム分離型EXEファイルは、ランタイムモジュールを別のファイルとしていますから、コンパイルで生成されるプログラムのコードサイズは、独立型EXEファイルにくらべて小さくなります。

●独立型EXEファイル

独立型EXEファイルは、コンパイルの際にランタイムモジュールをプログラムの一部としてリンク（結合）してしまいます。

そのため、プログラムの実行時には、ランタイムモジュールを必要としませんが、コンパイルのときには、「BCOM45A. LIB」と「BCOM45E. LIB」という2つのランタイムモジュールが、Quick BASICと同じディレクトリにあるか、パスを設定したディレクトリになければなりません。

プログラムにランタイムモジュールをリンクするために、コンパイル後のコードサイズは、ランタイム分離型EXEファイルにくらべて大きくなりますが、実行時にランタイムモジュールの存在について注意する必要がないので便利です。

独立型EXEファイルは、他のプログラムに実行を引き継ぐ「CHAIN」ステートメントを使用した場合に、「COMMON」ステートメントで指定された変数を保存しないため、変数の内容を引き継ぐことができないので注意しなければなりません。

PART ○ 4

プログラミングの入口

行番号はいらない

上から下へと書かれた順に実行されていく

BASICのプログラムは、最も小さな単位として「行」を単位に構成されています。いままでのBASIC、例えば「N_{88}－日本語BASIC (86)」では、1行が

[行番号] [ステートメント] [：別のステートメント]

で構成されています。

Quick BASICのプログラムの行は、

[ステートメント] [：別のステートメント]

で構成されています。

行と行番号

「行番号」というのは、BASICプログラムの実行順序を決める番号で、プログラムの1行ごとの先頭に整数でつけるものです。

プログラムはこの行番号の順に実行され、「GOTO」ステートメントや「IF～THEN」ステートメントで次の実行順序を示すとき、プログラムの分岐先をはっきりさせるための目印でもあります。

行番号は、Quick BASICが現れるまでのいままでのBASIC、例えばPC－9801シリーズに標準で付属してくる「N_{88}－BASIC」や「N_{88}－日本語BASIC (86)」などのようなBASICでは、絶対に必要でした。

ところがQuick BASICでは、プログラムを書くために行番号の必要はありません。Quick BASICでは、どうして行番号が必要ないのでしょうか。どうやって、BASICにプログラムの実行順序を知らせるのでしょうか。

左の「順次処理プログラム」リストは、同じ簡単なプログラムを、いま

■順次処理プログラム

N_{88}－日本語BASIC (86)

```
10 INPUT A
20 INPUT B
30 C=A+B
40 PRINT C
50 END
```

Quick BASIC

```
INPUT A
INPUT B
C=A+B
PRINT C
END
```

■ IF～THEN分岐プログラム

N$_{88}$－日本語BASIC（86）

```
10 INPUT A
20 INPUT B
30 IF B=0 THEN GOTO 40 ELSE GOTO 60
40 PRINT "0で割れません"
50 GOTO 80
60 C=A/B
70 PRINT C
80 END
```

Quick BASIC

```
INPUT A
INPUT B
IF B=0 THEN
    PRINT "0で割れません"
ELSE
    C=A/B
    PRINT C
END IF
END
```

までのBASICと、Quick BASICで書いたものです。

Quick BASICには、行番号がありませんが、いままでのBASICと同じように実行されます。

Quick BASICのプログラムは、書かれた順番に上から下へと実行されるからです。

一方、N$_{88}$－日本語BASIC（86）も、ここでは同じように上から下にプログラムを実行しているだけですが、やはり行番号が必要です。

では、今度は左のリストはどうでしょうか。

条件によって実行する順番を変えていますが、どちらのプログラムも同じ動作をします。

Quick BASICでは、「IF～THEN～END IF」ステートメントのなかに実行するプログラムが入ってしまっています。

N$_{88}$－日本語BASIC（86）では、「GOTO」ステートメントを使って、実行する行を変えています。

これは一例ですが、プログラムの実行順序を変えるのも、Quick BASICでは、プログラムの書き方しだいで、行番号を使わずに、読みやすくできるようになっています。

ラベル

Quick BASICでは、行番号を使わずにプログラム実行の分岐ができますが、どうしてもプログラムの離れたところへ実行を移したい場合には、どうしたらいいでしょう。

いままでのBASICにもありましたが、こういう場合は「ラベル」を使います。

N_{88}－日本語BASIC（86）でラベルを表わすには、「*」（アスタリスク）をつけてからラベル名をつづけます。

Quick BASICでは、ラベル名のあとに「：」（コロン）をつけるのです。

どちらもBASICのステートメントやコマンドは、ラベルの名まえに使うことができません。

■ GOSUB～RETURN分岐プログラム

N_{88}－日本語BASIC（86）

```
10 INPUT A
20 INPUT B
30 GOSUB *KAKE
40 PRINT C
50 STOP
  ⋮
100 *KAKE
110 C=A*B
120 RETURN
```

Quick BASIC

```
INPUT A
INPUT B
GOSUB KAKE:
PRINT C
STOP
  ⋮
KAKE:
C=A*B
RETURN
```

Quick BASICでは、行番号つきのプログラムも実行することができます。

Quick BASICでは、行番号をラベルと同じように、プログラムの行を表わす「名まえ」として扱っています。

■行番号つきの順次処理プログラム

N88－日本語 BASIC (86)

```
10 INPUT A
20 INPUT B
30 C=A+B
40 PRINT C
50 END
```

Quick BASIC

```
40 INPUT A
10 INPUT B
20 C=A+B
50 PRINT C
30 END
```

Quick BASIC のプログラムでは、このように行番号がバラバラにつけられていても、プログラムが上から下へと順に実行されます。

Quick BASIC のプログラムでの行の構成をもう少し詳しく見てみると、

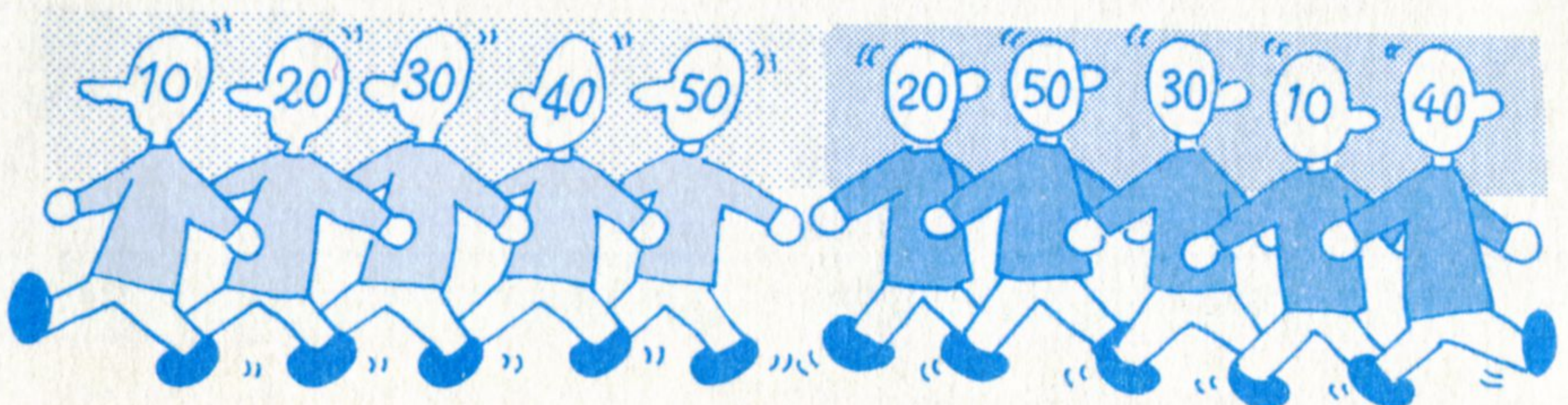
[行識別子][ステートメント][：別のステートメント][’コメント]

というように構成されています。

「行識別子」は、行番号とは違って、ラベルを含むプログラムの行につけられた名まえですから、なくてもかまいません。

Quick BASIC では、この行識別子に行番号やラベルを使えることになっています。

このようにして、いままでの BASIC で作成した行番号つきのプログラムも、Quick BASIC で動かすことができるようにしているのです。

「コメント」は、実行されない注釈で、「REM」ステートメントに相当するものですから、必要なときに入れます。

プログラムを構造化する

あとで直しや追加もしやすいように

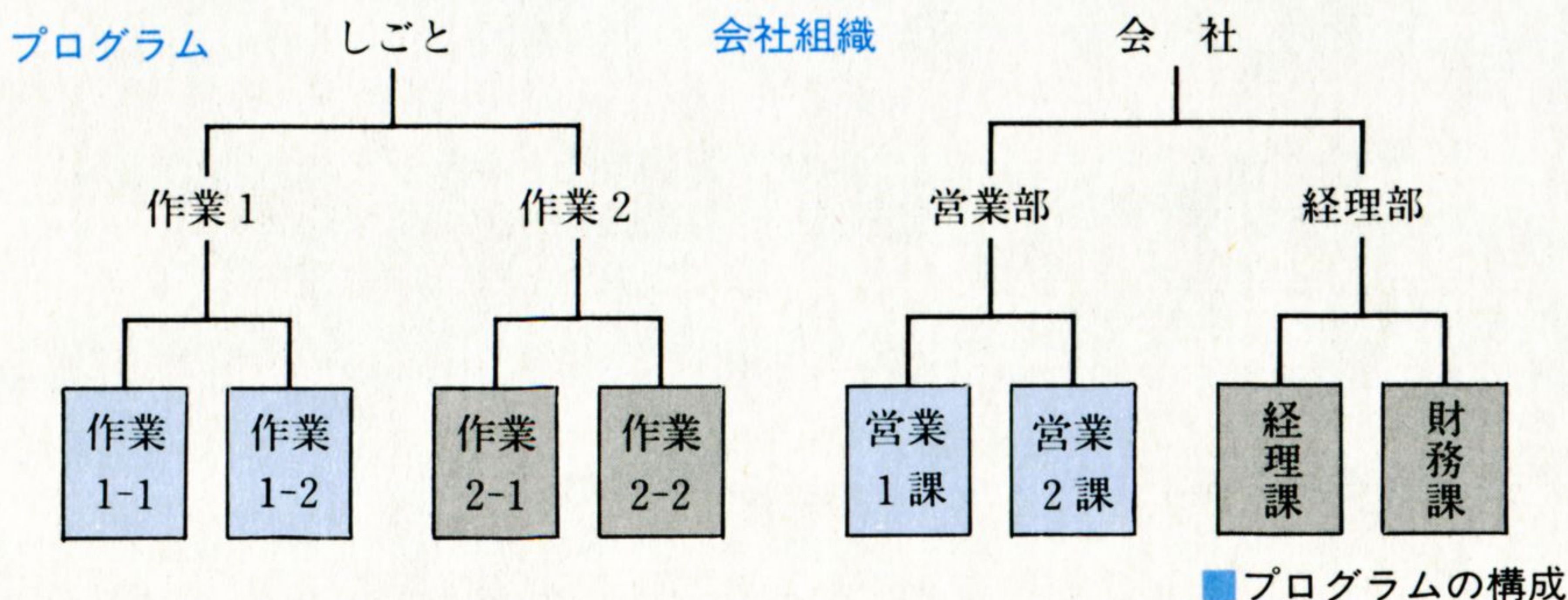
プログラムは、わかりやすく、わかりやすくと、それを意識して作成しないと、つくっているそのときこそ、なにをしているプログラムか、十分に理解しているものの、それから少し時間がたってしまうと、作成した本人でもわからなくなってしまって、プログラムを修正したり発展させたりするときに困ってしまう、などはよくあることです。

そんなことのないように、あとで見ても、そこにはなにが書いてあるか、どんな構造になっているかがひと目でわかるような、管理され、整備されたプログラムが、構造化されたプログラムです。
「プログラムの構造化」などというと、たいへん難しそうに聞こえますが、そんなことはありません。プログラミングするときの、ひとつの基本的な考え方なのです。

構造化プログラミング

プログラムの構造化は、プログラムを「書きやすく」「読みやすく」「理解しやすく」するための方法です。

構造化にはまず、プログラムを「ブロック」に分けてつくっていく方法があります。

ブロックは、ひとつの作業をするひとかたまりのプログラムで、大きなしご

■プログラムの構成

とをするためにいくつかのブロックを呼び出して、処理を行っていきます。

プログラムは、まず大まかなしごとを考えて、次にそのしごとをするための細かい作業をいくつか考えていきます。

こうすると、プログラムは、「ツリー状」にできあがっていきます。ちょうど、会社の組織のように構成されていることがわかります。

このようにプログラムの構成を上から下へと、だんだん細かくしていく方法で、大きなプログラムの流れをつかみながら、細かいプログラムを正確につくっていくことができます。

もしプログラムが意図したとおりに動作しなくても、このように管理されていれば、不都合が生じた部分をすぐに発見して、対策を立てることができます。

また、ある程度プログラムができあがってきて、新たな機能をつけ加えたりする場合には、逆に下から上へとプログラムを検討していくこともできます。

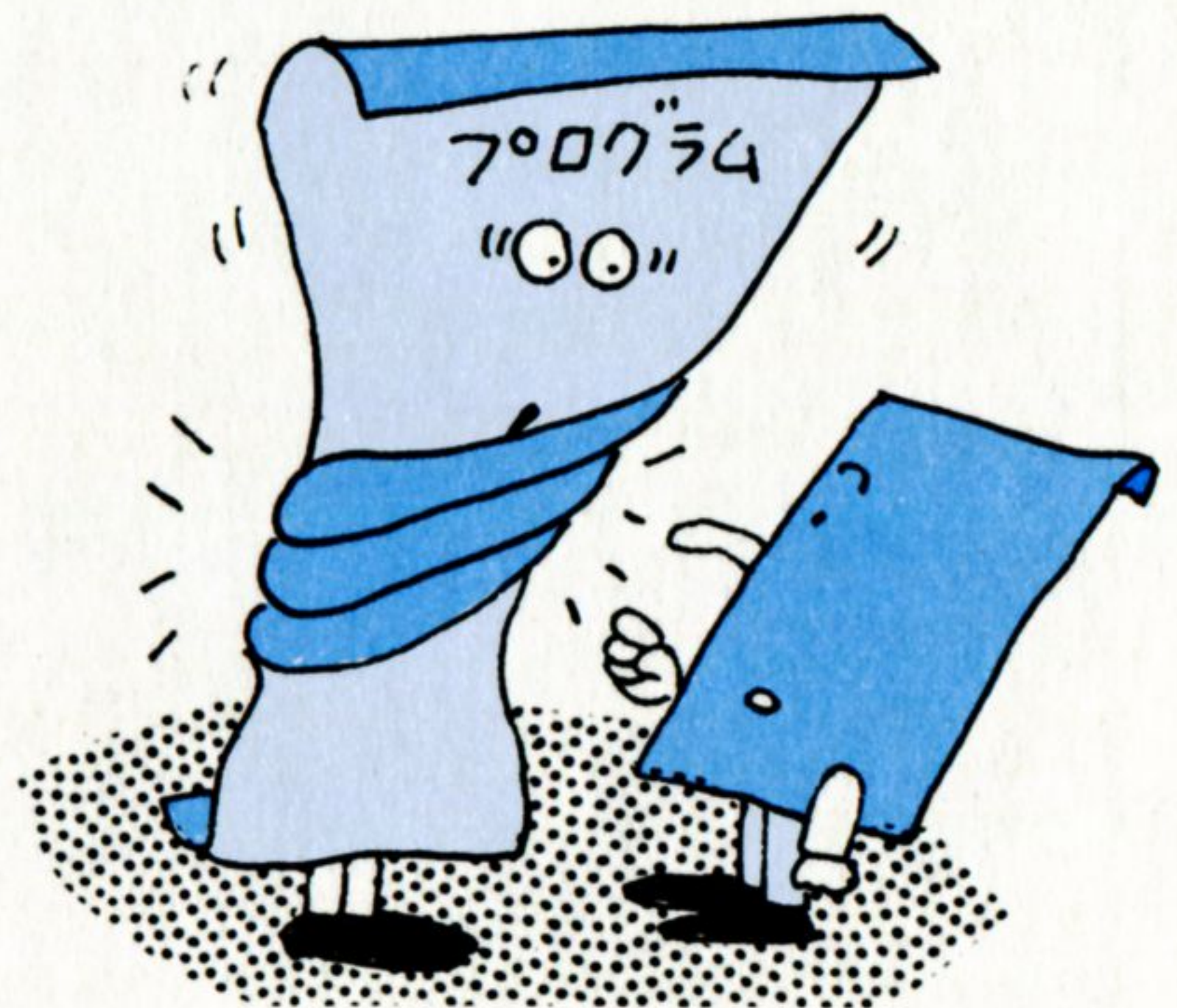
常にプログラムの構造化を意識していれば、自然に上から下へ実行されていくわかりやすいプログラムができますが、これとは逆に、プログラムをつくるときに気をつけなければならないことや、やってはいけないこともあります。

● GOTOでスパゲッティ

GOTOステートメントは使わないようにします。

特に、IF~THENステートメントと組み合わせて、プログラムの実行をバラバラに分岐させると、「スパゲッティプログラム」といわれる、実行順序のめちゃくちゃな、わかりにくいプログラムになってしまいます。

● マルチステートメント

マルチステートメントとは、複数のステートメントを1行のプログラムに押し込んでしまうことですが、関連の深いステートメントを1行にまとめることはよいとしても、いくつものステートメントを「：」（コロン）でつないでいくことも、わかりにくいプログラムになる原因です。

プログラムの制御構造

Quick BASICでは、行番号を使わないで、構造化されたプログラムをつくるために、プログラムの制御構造は使いやすく、わかりやすいものになっています。

プログラムの構造化のための基本的な制御構造は、

順次処理

選択

繰り返し

の3つの要素で構成されています。

すべてのプログラムは、この3つの制御構造を使って作成することができます。

順次処理

順次処理は、上から下へと順番に実行していく構造で、行番号がなくても下の左の図のように順番に実行していくことができます。

しごと1
↓
作業1－1
↓
作業1－2
↓
作業1－3

しごと2
↓
作業2－1
↓
作業2－2
↓
作業2－3

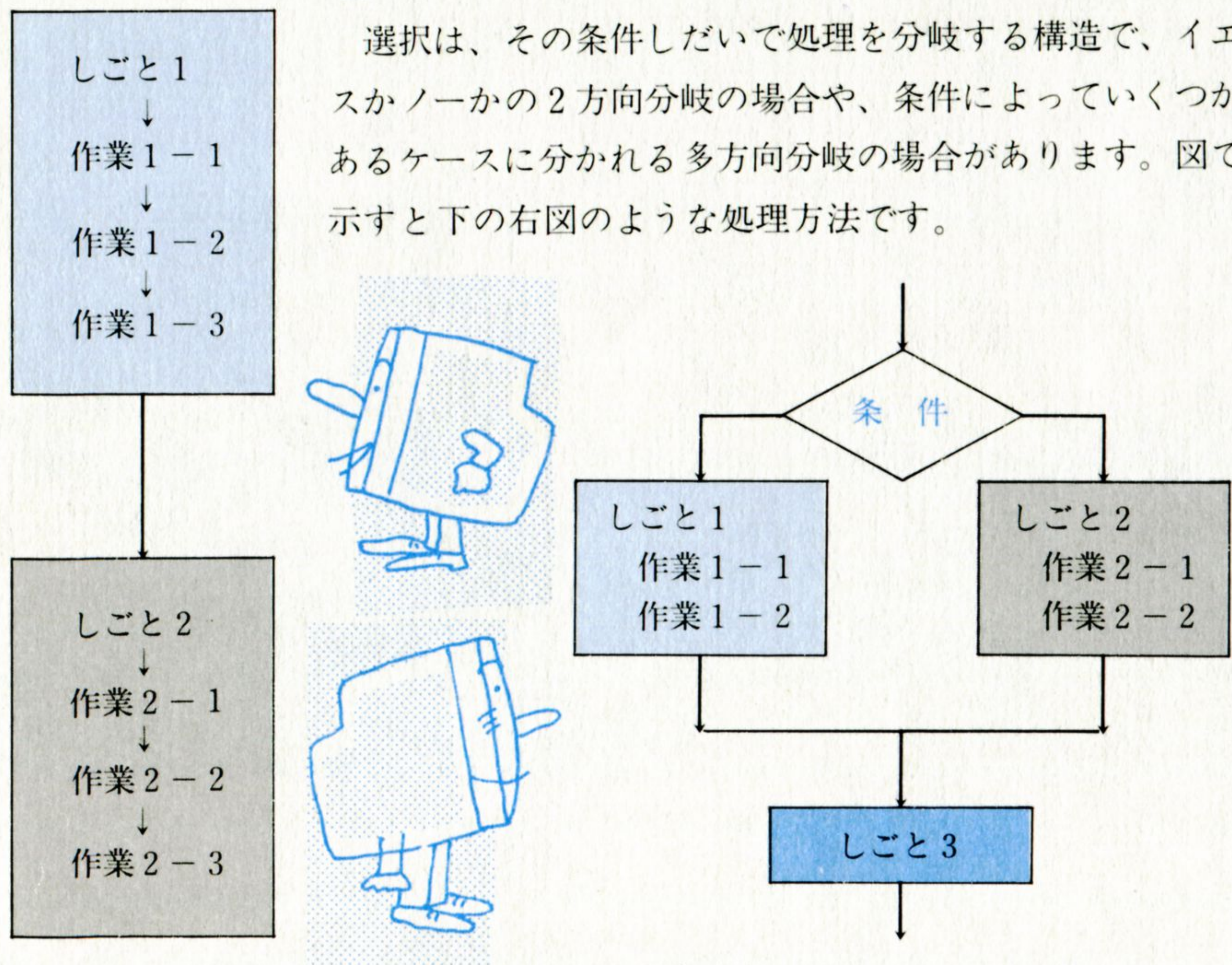
選択

選択は、その条件しだいで処理を分岐する構造で、イエスかノーかの2方向分岐の場合や、条件によっていくつかあるケースに分かれる多方向分岐の場合があります。図で示すと下の右図のような処理方法です。

条件

しごと1
作業1－1
作業1－2

しごと2
作業2－1
作業2－2

しごと3

繰り返し

繰り返しは、条件によって同じ処理を何度も繰り返す（ループ）構造です。条件判断を繰り返しの前に行う場合と、あとで行う場合があります。

条件判断を繰り返しの前に行うと、判断の結果によっては一度も処理を行わずにループを抜けてしまう場合もありますし、あとで判断を行うと、必ず１回はループ内の処理を実行することになります。

前判断型

ループする？

ノー

条　件

イエス

しごと１
　作業１－１
　作業１－２

しごと２

後判断型

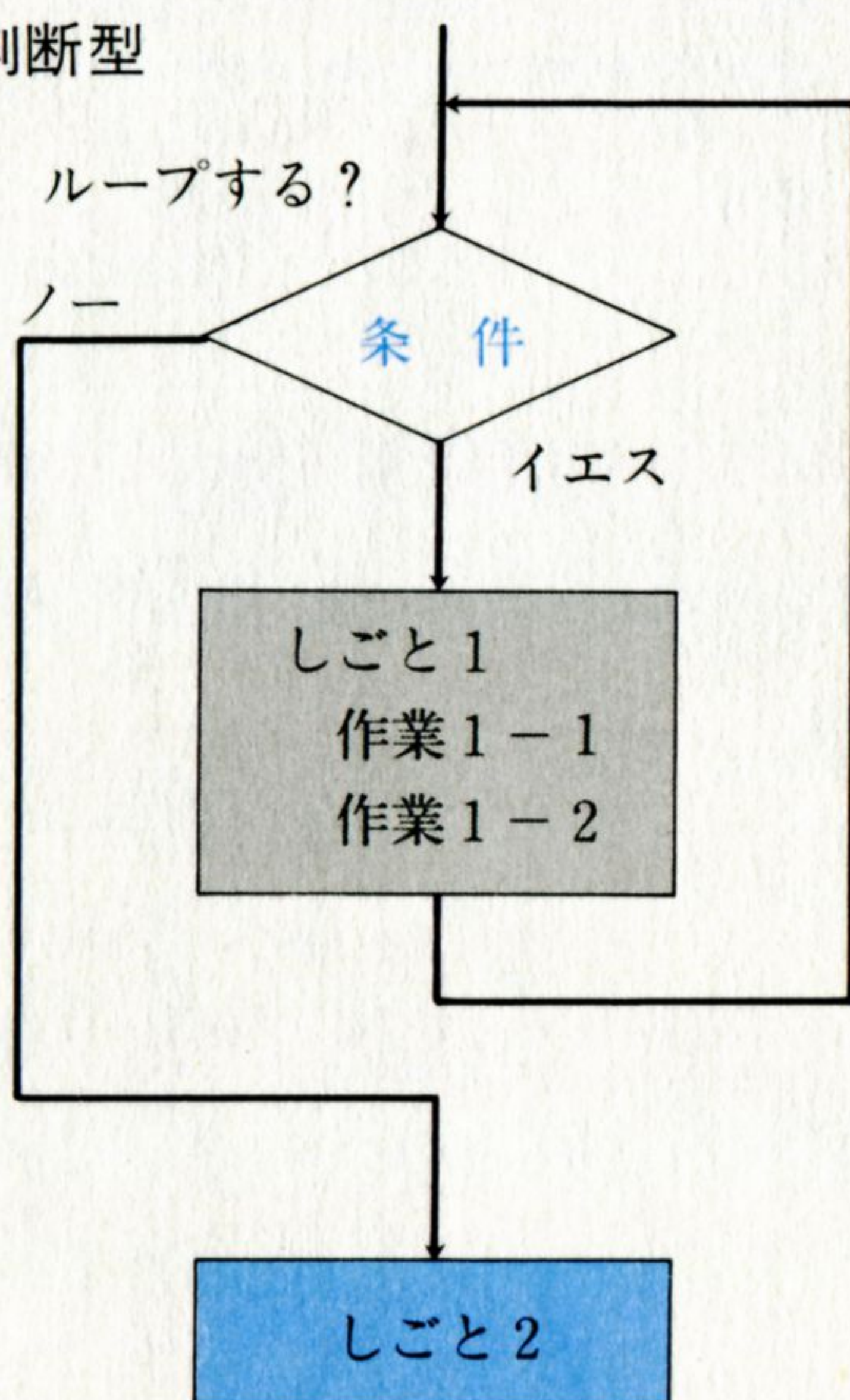
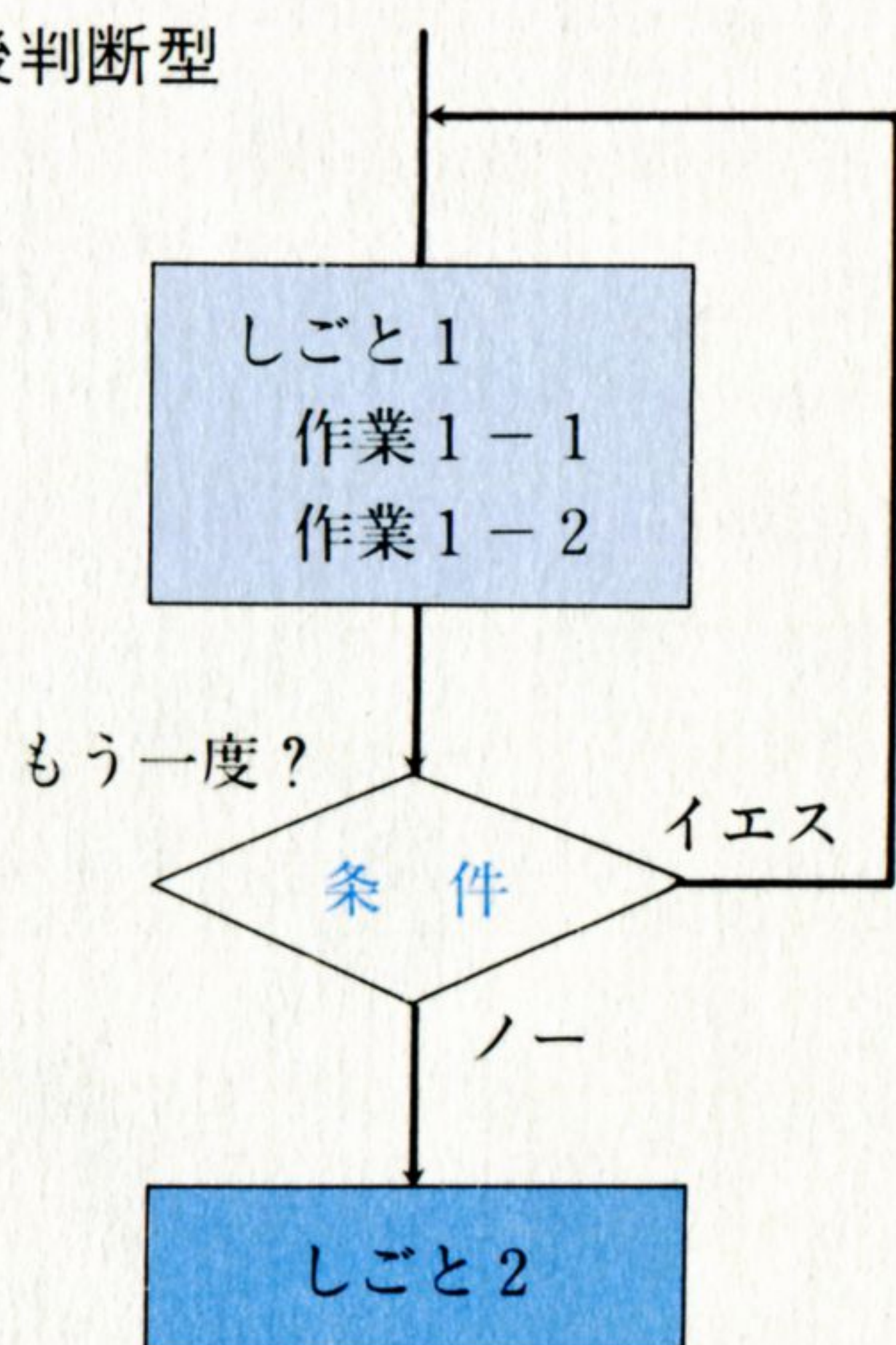
しごと１
　作業１－１
　作業１－２

もう一度？

条　件

イエス

ノー

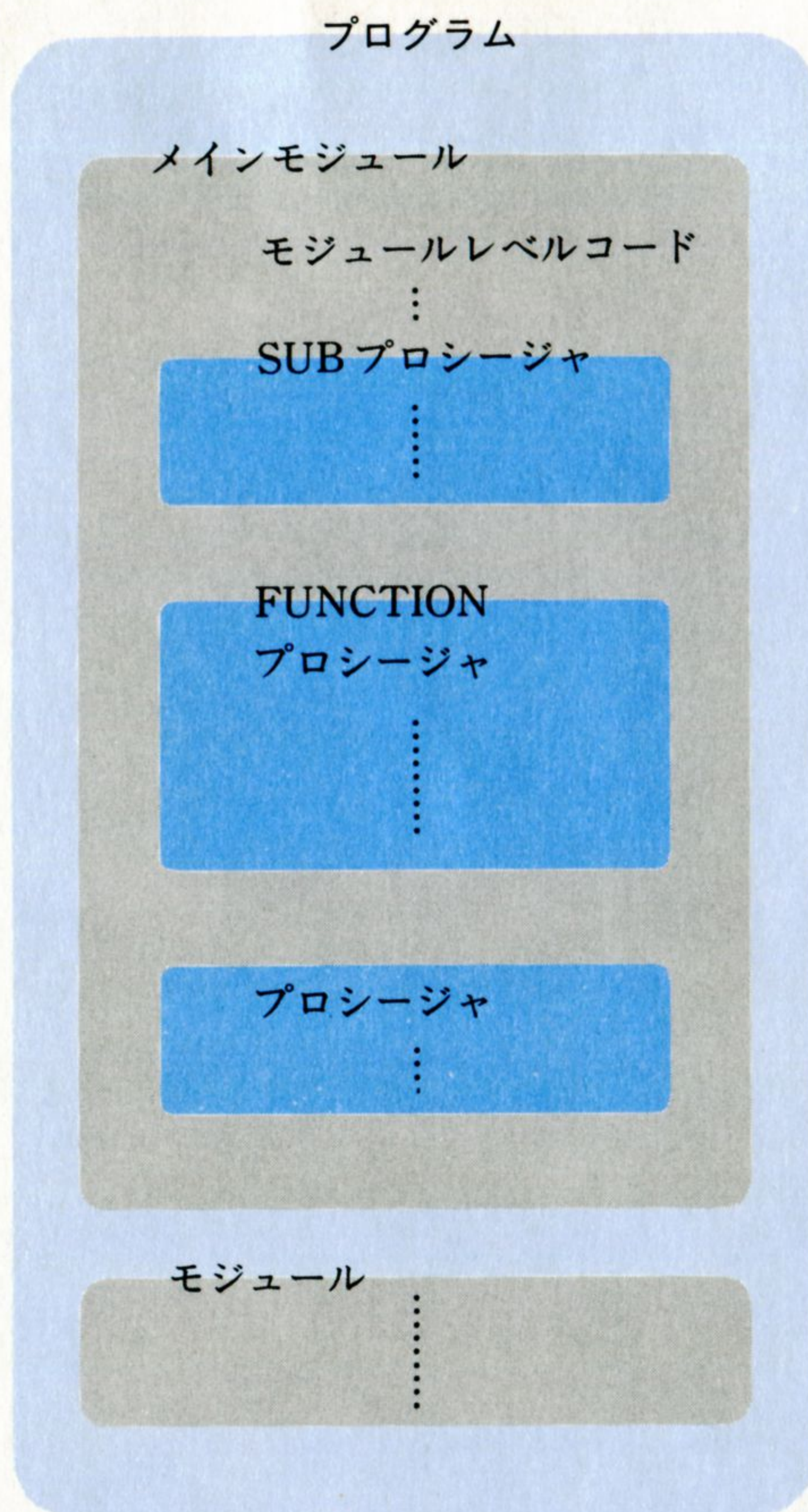
しごと２

プロシージャ

Quick BASICでは、いままでのBASICと同じように「GOSUB～RETURN」ステートメントによるサブルーチンを利用してプログラムを構造化することもできますが、「プロシージャ」を使えば構造化をさらに進めることができます。

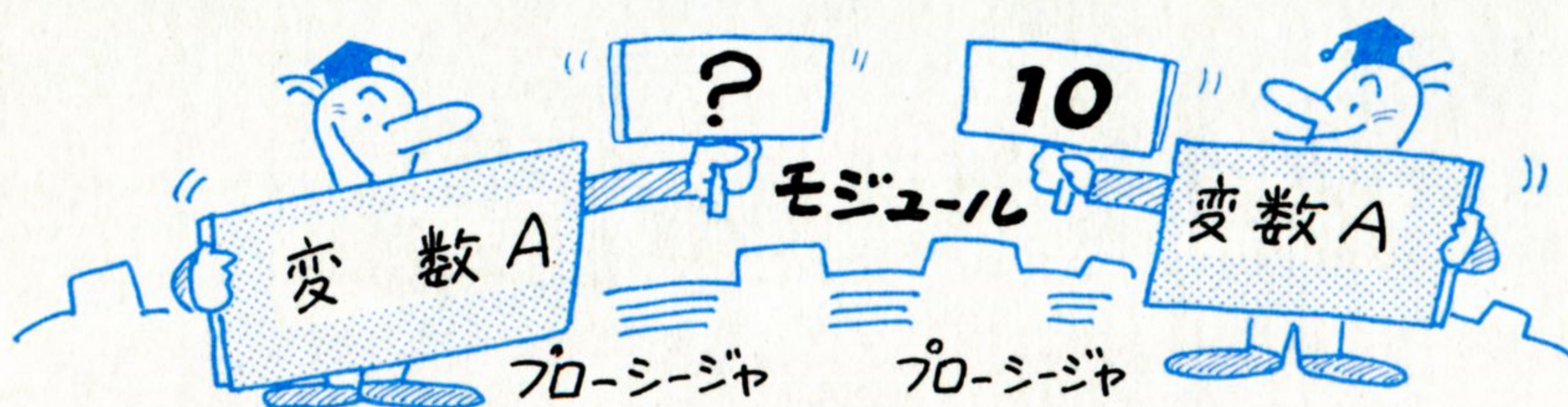
プロシージャは、ひとつのまとまったしごとを行うプログラムの単位で、本体（メイン）のプログラムとは独立したひとつの「モジュール」として管理されます。

メインモジュールをひとつのプログラムとして見てみると、図のように、「モジュールレベルコード（メインモジュール）」と「プロシージャ」から構成されています。

プログラムとして独立しているだけではなく、変数も独立して使用できるようになっています。

プロシージャ内では、「グローバル変数」に指定しないかぎり、すべての変数が「ローカル変数」として独立していますので、モジュールレベルコード（メインモジュール）や他のプロシージャで使われているのと同じ変数名を使用しても、他のプロシージャにはまったく影響を与えません。

■プログラムの構成

他のプロシージャと同じ変数名も使える

グローバル変数とは、いままでのBASICの変数と同様に、プログラム全体で共通に使用される変数で、特に宣言すれば、異なったプロシージャで変数の値を参照したり、代入することができるようになります。

変数
ローカル変数　引数
グローバル変数　パラメータ

これに対してローカル変数は、特定のプロシージャ内だけに有効な変数で、他のプロシージャで同じ名まえの変数を使っていたとしてもお互いに影響されることはありません。

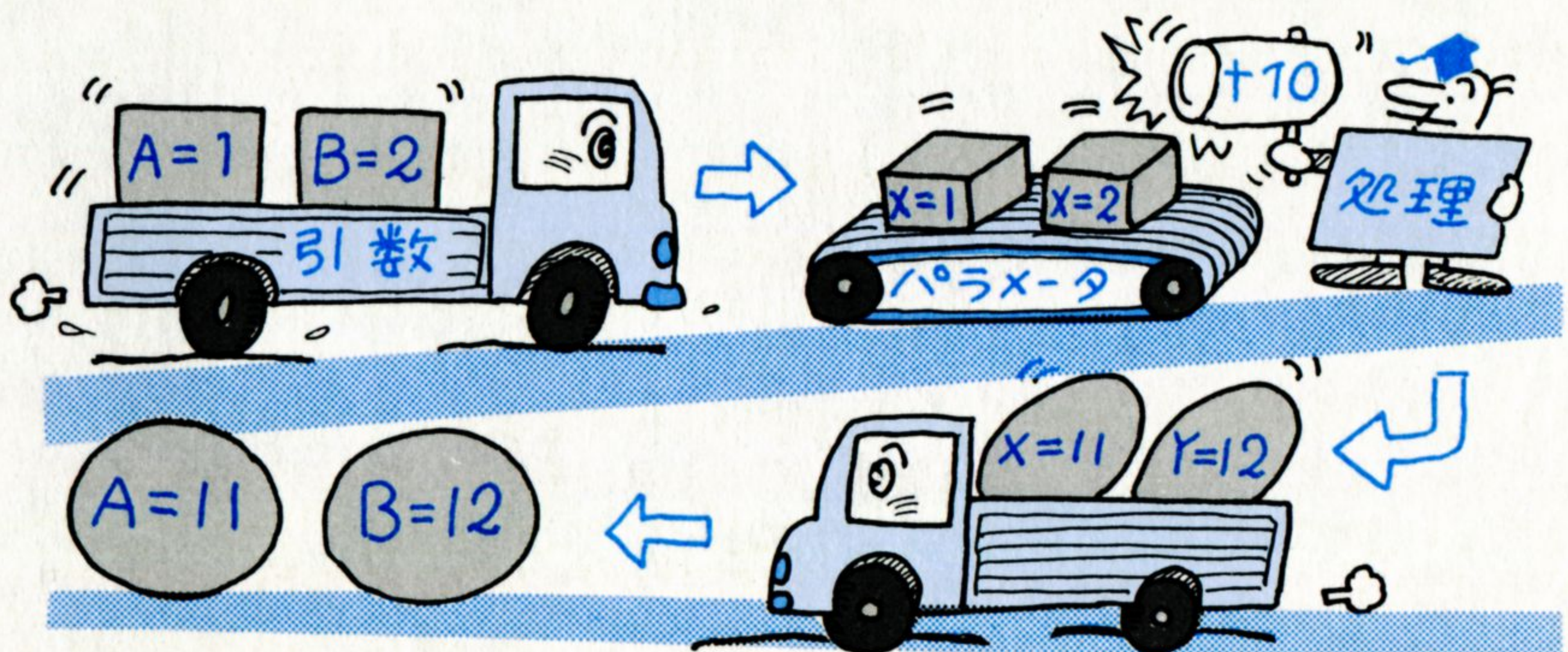
メインプログラムと違った変数で処理できる

プロシージャの汎用性を高めるために、「パラメータ」を使用して、メインプログラムから処理すべきデータを「引数」としてプロシージャに一括して引き渡し、メインプログラムとプロシージャで異なった変数の組による処理をすることができます。

プロシージャを呼び出すときに、値を引数としてプロシージャに引き渡すと、その値によってプロシージャ内での処理を行います。

プロシージャ内で処理が終わると、プロシージャを呼び出したステートメントの次のステートメントに戻りますが、このとき呼び出したときに使われたメインプログラムの変数に、処理を終わったプロシージャの変数の内容である処理結果の値を返します。

パラメータは、プロシージャで使われる変数の値を受け取る仮の変数の「名まえ」で、プロシージャ内でだけ有効です。

引数は、パラメータに引き渡される変数の内容である「値」で、プロシージャでの処理が終わると、パラメータに引き渡した変数に値が戻されます。

引数リストとパラメータリストの変数名は、同じでなくてもかまいませんが、引数とパラメータの「個数」と「変数の型」は、同じでなければなりません。

プロシージャは、ひとつの独立したプログラムの単位になっているので、メインモジュールからだけではなく、他のプロシージャや他のモジュールから呼び出すことができます。

また、一度作成したプロシージャは、他のプログラムに組み込んで使うことができるので、同じようなプロシージャをプログラムごとに作成しなくてもすむようになり、プログラミングの省力化や、グループでのプログラム開発ができるようになります。

プロシージャには、「SUB（サブ）プロシージャ」と「FUNCTION（ファンクション）プロシージャ」の2種類があります。

SUBプロシージャ

SUBプロシージャは、GOSUBステートメントに似た働きをします。

SUBプロシージャが呼び出されると、プロシージャ内のステートメントが順に実行され、「END　SUB」ステートメントにくるとSUBプロシージャを呼び出したステートメントの次のステートメントに実行を戻します。

● SUBプロシージャの定義

SUBプロシージャは、

```
SUB [プロシージャ名] [パラメータリスト]
 [SUBの処理]
     :
     :
EXIT SUB
     :
     :
END SUB
```

のように行われ、初めのSUBステートメントを入力すると、自動的にSUBプロシージャの定義に入り、Quick BASICの管理しているサブファイルのリストに加えられて、メインのモジュールと区別されます。

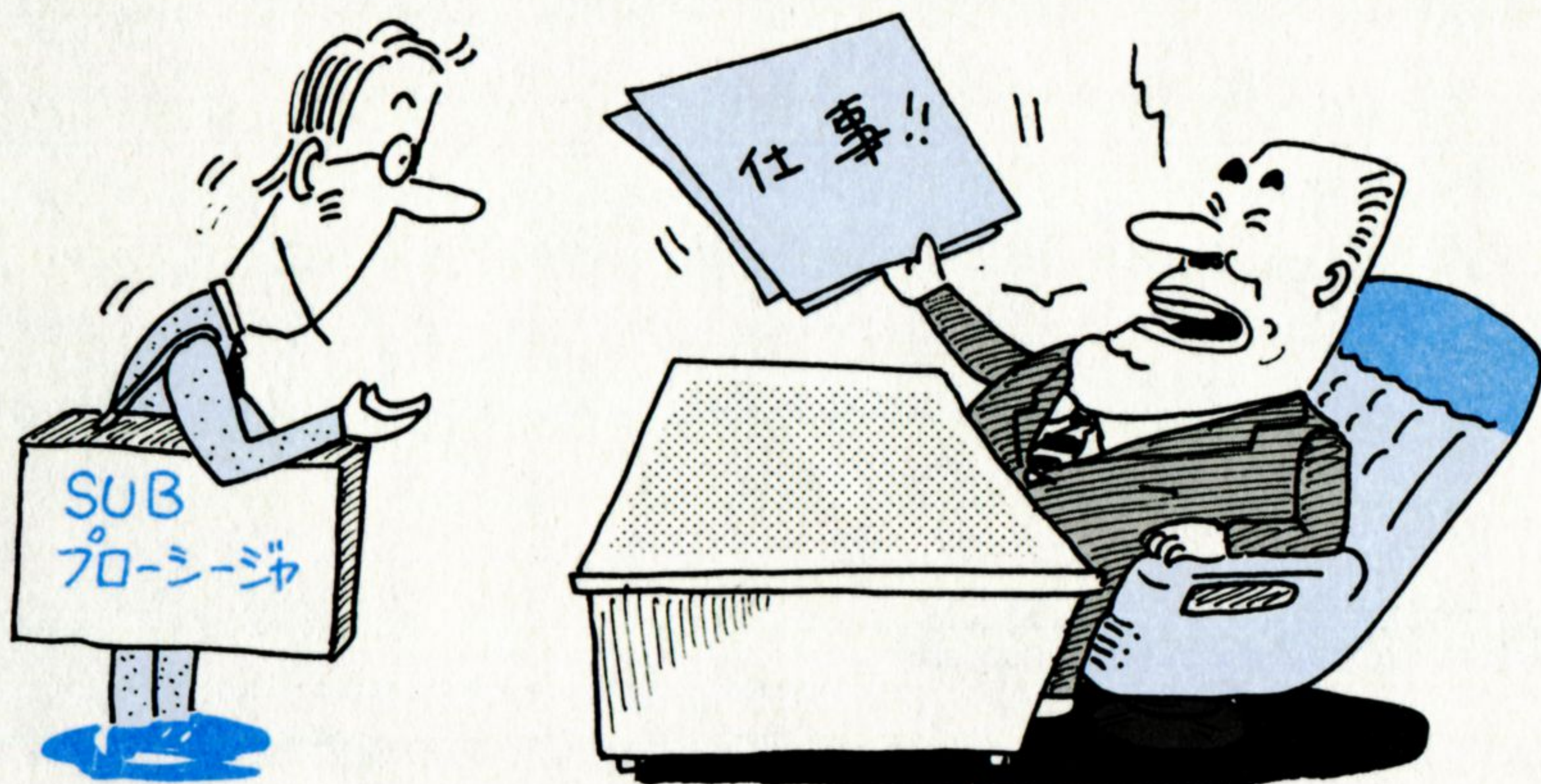
［プロシージャ名］は、先頭が「FN」で始まらない40文字までの任意の名まえをつけられます。ただし、Quick BASICの予約語※は使用できません。

［パラメータリスト］は、変数のリストです。この変数は、SUBプロシージャを呼び出したモジュールからの引数を受け取るSUBプロシージャ内で有効なパラメータです。

SUBプロシージャの末尾は、「END SUB」で終わり、SUBプロシージャの処理を終了して、呼び出されたステートメントの次のステートメントに実行を戻します。

また、SUBプロシージャの途中で処理を中断して、呼び出したプログラムに処理を戻すためには、「EXIT SUB」ステートメントを組み入れます。

● SUBプロシージャの呼び出し

SUBプロシージャは、「CALL」ステートメントに組み込んで呼び出す方法と、プロシージャ名をQuick BASICのステートメントと同じようにそのまま使って呼び出す2通りの方法があります。

※予約語　Quick BASICのステートメントやコマンド、システム関数として使用されている単語。

Quick BASICがプログラムを実行するときに、単語を1語ずつ読み込んで解釈していきますが、このときにまずQuick BASICのステートメントなどであるかどうかを調べます。

Quick BASICのステートメントなど以外の場合は、プロシージャ名や変数名として処理をするので、予約語を変数などの名まえに使用すると、Quick BASICのステートメントなどとして実行しようとするために、エラーになってしまいます。

CALL [SUB プロシージャ名] [(引数リスト)]

CALLに組み込まれて[SUBプロシージャ名]で指定されたSUBプロシージャが呼び出されます。

[(引数リスト)]は、SUBプロシージャに引き渡す引数の入った変数を「( )」(かっこ)でくくったなかに、「,」(カンマ)で区切って指定します。

[SUB プロシージャ名] [引数リスト]

Quick BASICの組み込みステートメントと同様に、[SUBプロシージャ名]だけをプログラムに書くと、指定されたSUBプロシージャが呼び出されます。

[引数リスト]は、SUBプロシージャに引き渡す引数の入った変数を「,」(カンマ)で区切って指定します。

この場合、引数リストを「( )」(かっこ)でくくってはいけません。

SUBプロシージャ名だけで呼び出す場合は、プログラムの先頭で「DECLARE」ステートメントを使って、SUBプロシージャの使用を事前に宣言しておかなければなりません。

DECLARE SUB [SUB プロシージャ名]

Quick BASICのエディタ環境でプログラムを作成しているときには、CALLに組み込まないSUBプロシージャの定義を行うと、Quick BASICがこの宣言を、自動的にモジュールの先頭で行います。

このため、Quick BASICのエディタ環境でプログラムを作成しているときには、SUBプロシージャを定義するだけで、特に宣言を意識する必要はありませんが、他のエディタなどを使ってプログラムを作成している場合は、必ず自分でこの宣言を行わなければなりません。

FUNCTIONプロシージャ

FUNCTIONプロシージャは、「DEF FN関数定義」に似た働きをしますが、独立したプロシージャですので、FN関数より自由に使うことができます。

FUNCTIONプロシージャは、呼び出すプログラム内ではQuick BASICの関数と同じように、式のなかで使います。

式のなかにFUNCTIONプロシージャ名が現れると、FUNCTIONプロシージャが呼び出され、プロシージャ内のステートメントが実行されます。「END FUNCTION」ステートメントに来ると、呼び出された式に処理を戻し、処理の結果を関数の値として、呼び出した式の計算や評価をつづけます。

●FUNCTION プロシージャの定義

FUNCTION プロシージャの定義は、

```
FUNCTION [プロシージャ名]([パラメータリスト])
  [FUNCTION の処理]
      :
EXIT FUNCTION
      :
  [プロシージャ名]= [処理結果]
END FUNCTION
```

のように行われ、初めの「FUNCTION」ステートメントを入力すると、自動的にFUNCTION プロシージャの定義に入り、Quick BASIC が管理しているサブファイルのリストに加えられて、メインのモジュールと区別されます。

[プロシージャ名] は、先頭が「FN」で始まらない40文字までの任意の名まえをつけることができます。ただし、Quick BASIC の予約語は使えません。

[パラメータリスト] は、FUNCTION プロシージャに引き渡す引数の入った変数を「( )」(かっこ) でくくったなかに、「,」(カンマ) で区切って指定します。

[プロシージャ名]= [処理結果] は、プロシージャ内で処理した結果の格納されている変数の値を、プロシージャ名に代入すれば、FUNCTION プロシージャを呼び出したプログラムに値が返されます。

[プロシージャ名] に値を代入しないと、初期値である数値関数では「0」、文字関数では「ヌル」(空(から)の文字列) を結果として返します。

[処理結果] は、プロシージャ内で処理された結果を格納している変数です。

FUNCTION プロシージャの末尾は、「END FUNCTION」ステートメントで終わり、FUNCTION プロシージャの処理を終了して、呼び出された式に実行を戻します。

また、FUNCTION プロシージャの途中で、処理を中断して戻すためには、

「EXIT FUNCTION」ステートメントを組み入れます。

● FUNCTION プロシージャの呼び出し

FUNCTION プロシージャは、「INT」関数などの Quick BASIC の組み込み関数を使うのと同じようにして、呼び出します。

ただし、呼び出す FUNCTION プロシージャが違うモジュール内にある場合は、呼び出す側のモジュールで DECLARE ステートメントでプロシージャの使用宣言をしておかなければなりません。

プロシージャ名を式のなかで使うと、引数として引き渡された変数をプロシージャ内で処理して、この変数の値を結果として返し、式の計算をつづけます。

また、関数と同じように「型宣言文字」(%、&、!、#、$など) をつけ加えることで、名まえの示す型で返される値の型を指定することができます。

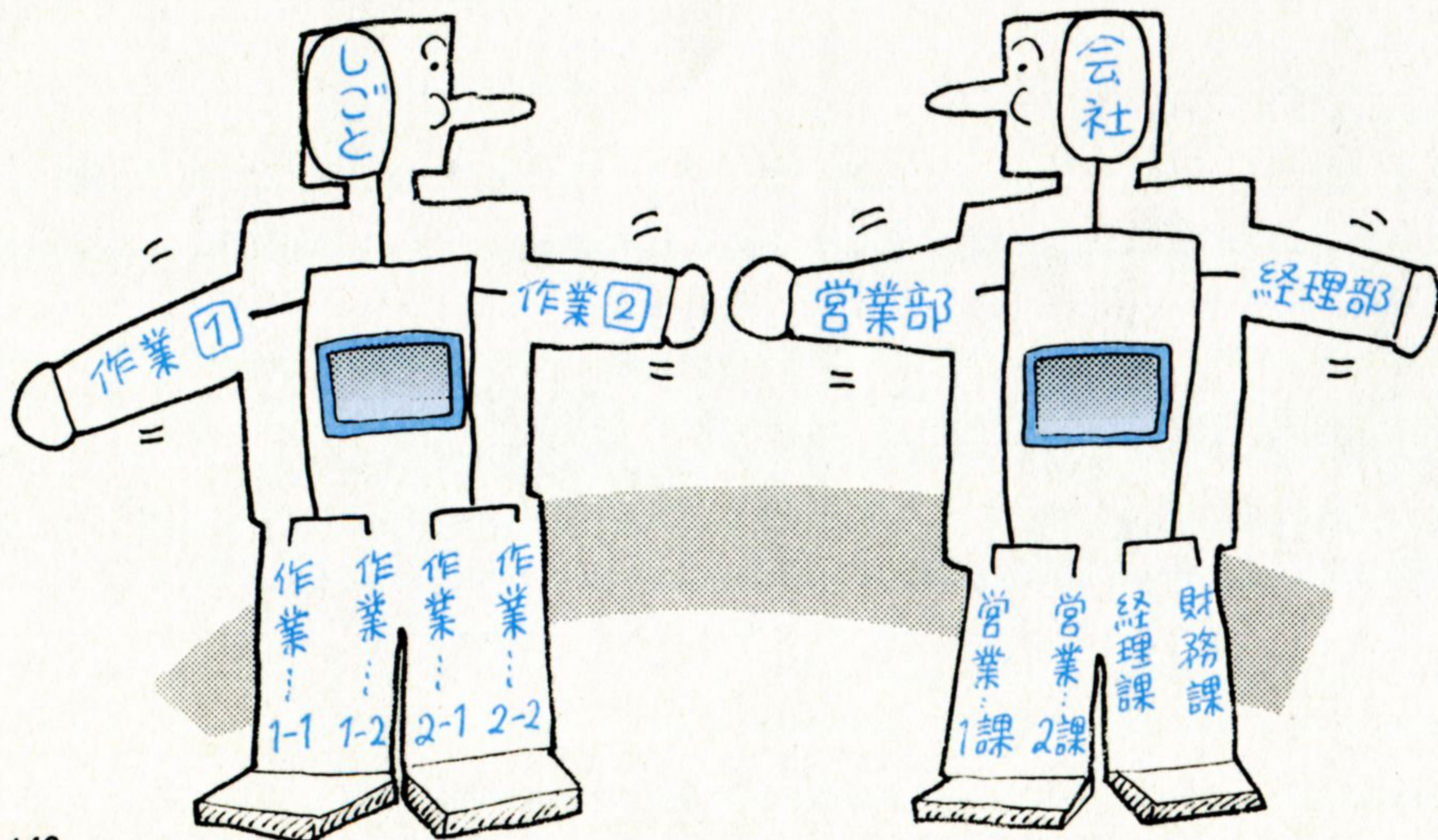
例えば、文字列の処理を行う FUNCTION プロシージャの場合は、

```
FUNCTION Moji$(m1$ , m2$)
    m3$＝m1$＋m2$
    Moji$＝m3$
END FUNCTION
```

のように、「$」(ダラー) をつけて定義します。

呼び出したプログラムの式のなかで計算がつづけられ、

```
Hidari$＝LEFT$(Moji$("Quick" , "Basic") , 7)
```

の場合は、変数「Hidari$」には、「QuickBa」という文字列が代入されます。

変数と代入の入口

変数はデータを出し入れする箱と考えよう

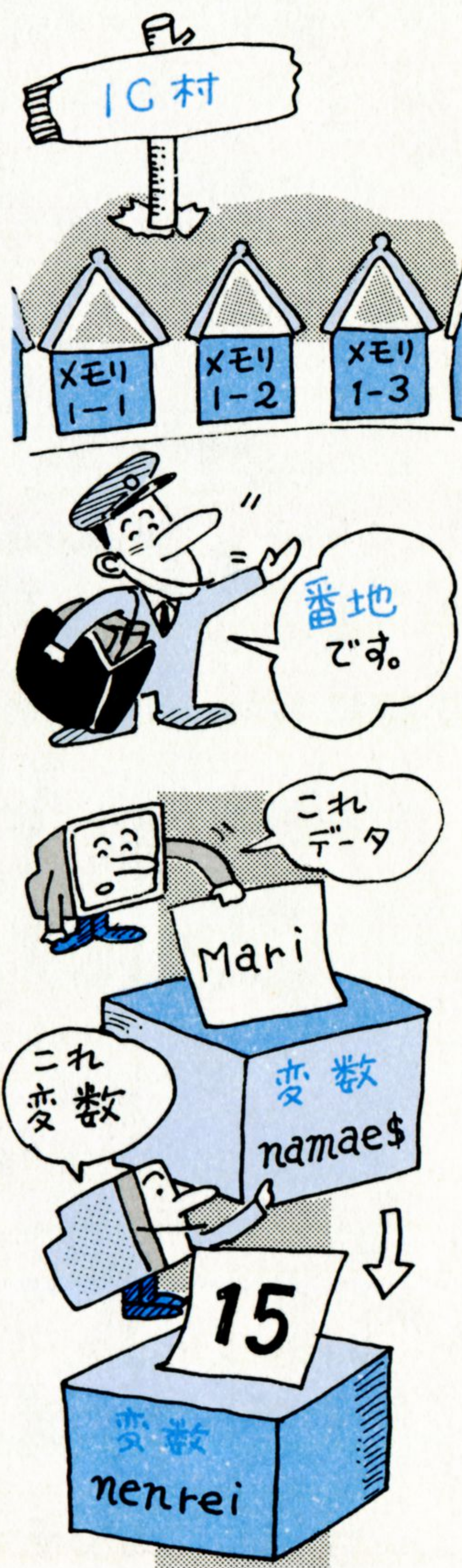
変数は、パソコンで処理をするためのデータを入れておく場所であり、変数にデータを入れることを代入といいます。

パソコンのなかでの実際のデータは、ICできたメモリのなかに0と1からなる2進法のデータの列として蓄えられます。メモリはひとつずつ番地が定められていて、データはこのメモリのなかに、順番に記録されています。

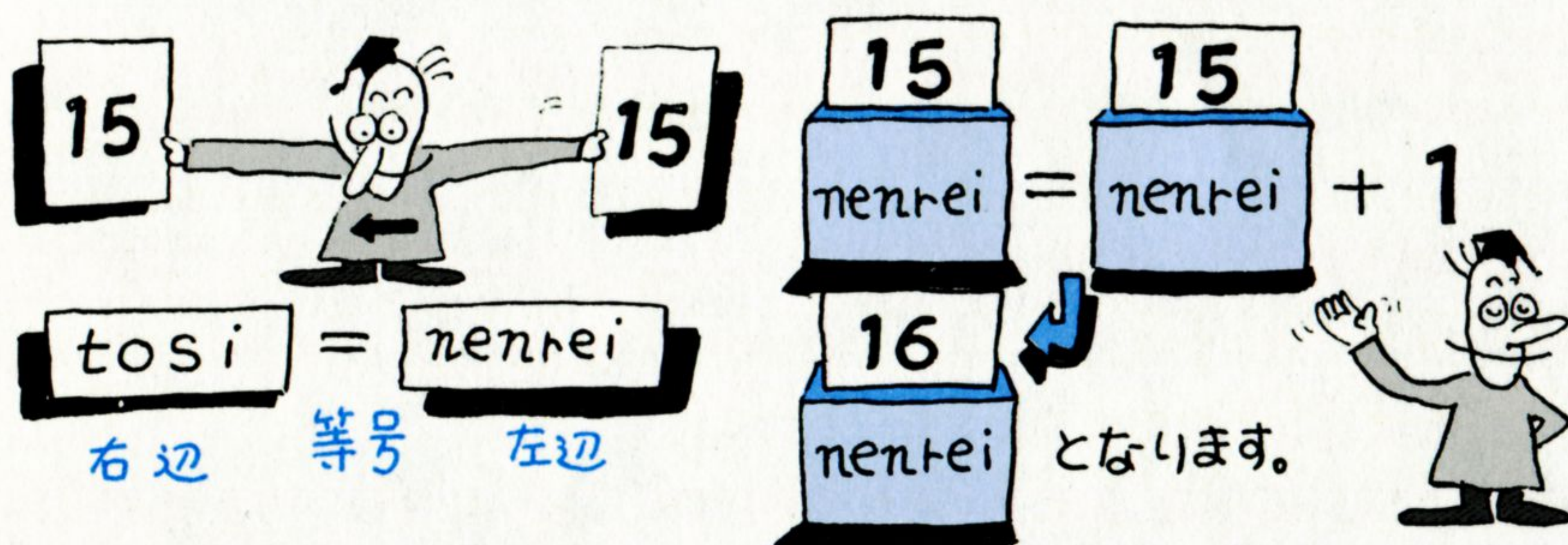
変数は、このデータのあるメモリの番地を管理しています。ある番地から始まる、ある長さのデータは、特定の変数名をつけられた変数の値（内容）として扱う約束になっています。

このようにメモリにある姿で、変数を理解するにはちょっと抵抗がありますが、本来はこのようなものであると知っておくと、より上級のプログラムを作成したり、C言語など他の言語を勉強するときに役立ちます。ここでは深くは触れませんが、頭の片隅に置いておいてください。

変数は、よく箱にたとえられます。箱のなかに、数値や文字などのデータを入れたり出したりするイメージで考えると、わかりやすいでしょう。

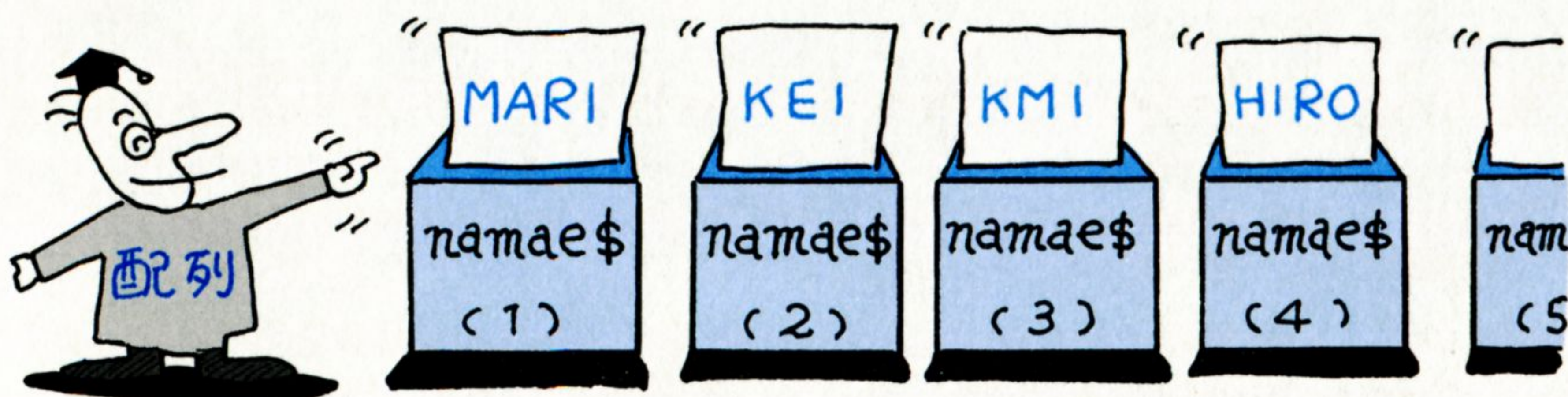
変数という箱のなかにデータをいれることを代入といいます。

Quick BASIC では、代入は「＝」（イコール、等号）を使い、等号の左辺の変数に、右辺のデータを代入します。データには、数値、文字列、式、変数がきます。

右辺の式は、数式の場合は計算され、条件式の場合は評価されて、左辺の変数に結果が格納されます。右辺に変数がきた場合には、右辺の変数に格納されている内容が、左辺に格納されます。

Quick BASIC では、右辺で変数を使った計算を行って、左辺の変数に結果を代入しますから、左辺、右辺に同じ変数を使うと、右辺の変数の値に計算を施して、また左辺の同じ変数に代入して、値を更新することができます。

変数に背番号をつけて管理する配列変数

変数には、配列変数という便利なものもあります。変数に背番号をつけたようなもので、同じ配列変数名に添字をつけて管理しています。

背番号1番の人の名まえは、配列変数 namae$(1)に、2番は namae$(2)に格納するというように使います。こうすると、FOR～NEXT ステートメントなどのループと組み合わせて、背番号を順番に指定するだけで、たくさんの変数をまとめて取り扱うことができるようになります。

配列変数を使うと、パソコン向きの大量のデータをすばやく次つぎに処理していくしごとができるのです。

しっかり変数管理

データの型で扱い方が違ってくる

データの型

パソコンでは、キーボードやフロッピーディスクから「データ」を入力して、計算や判断などの処理を行い、ディスプレイやフロッピーディスクにその結果を出力します。

処理された結果は、数値や文字ばかりではなく、グラフィックの絵やブザーの音の場合もありますが、これらもパソコンの内部では、同じような「データ」として処理されています。

プログラムは、このような入力装置から得たデータをいろいろと加工して、出力装置に取り出す手順を示しています。

Quick BASIC では、「データの型」の種類によって取り扱い方法が違っています。

データ型は、大きく「文字型データ」と「数値型データ」に分かれます。

文字型データは、1バイト文字※のアルファベットやカナ、2バイト文字※の日本語の漢字やひらがななどをさしています。

数値型データは、量や数などを表わす数値のことをさします。

● 文字と数字と数値

ここで注意してほしいのは、「数字」は文字であって、数値ではないということです。「1234」は、1,234という数や量を示す数値を表わす場合もありますし、文字として1234という4文字の文字列である場合もあります。

1234を文字型データとして扱う場合は、

```
MOJI$="1234"
```

と、「"」(ダブルクォーテーションマーク)で囲って、文字型データであることを示します。

また、数値型データの場合は、

```
KAZU=1234
```

と、そのまま表わします。

1234は数値型データであり、計算の対象とすることができます。

文字型データ

Quick BASICでは、文字型データとして32,767文字までの長さの文字列を取り扱うことができます。

文字型データは、次の2つの種類があります。

可変長文字列　文字列の長さが定められていない、最大32,767文字までの文字の並びです。

固定長文字列　文字列の文字数が宣言されて決められている文字列で、最大32,767文字までの範囲で宣言することができます。

固定長文字列の文字数の宣言は、

```
DIM Moji$ AS STRING * 10
```

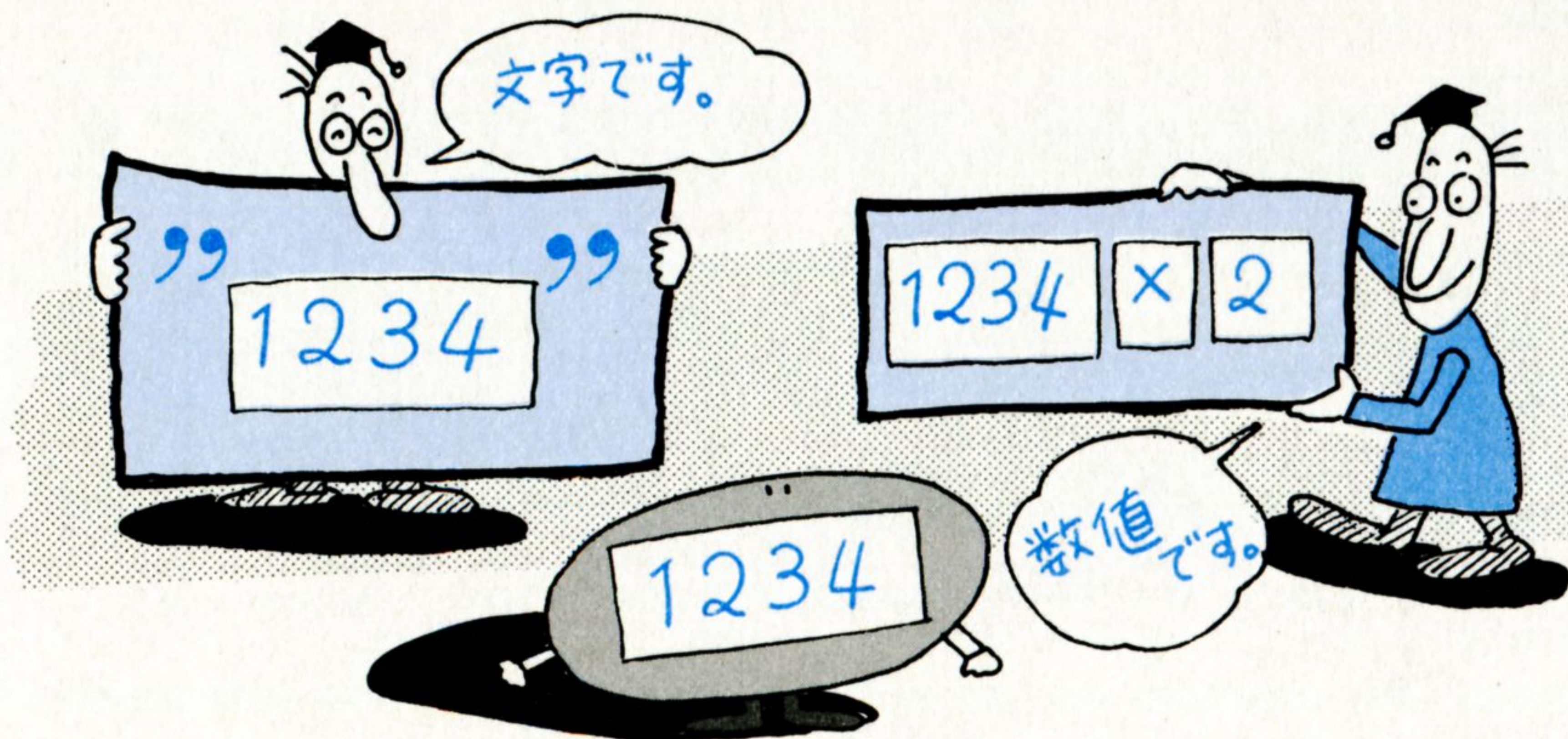
のようにDIMステートメントを使って、「Moji$」という変数を、10文字の固定長の文字型データを取り扱う変数として宣言します。

数値型データ

数値型データには、次の4つの型があり、順に精度が高くなっています。

精度が高くなるに従って、メモリに記憶されるときに使用する領域のバイト数が多くなります。精度の高い数値を、精度の低い変数に代入するなどの「型変換」を行うと、精度が低いほうの型に合わせられます。

また、浮動小数点形式の数値は、2進数で計算された結果を10進数に変換して表示していますので、計算誤差を含んでいます。金銭の計算や、計算結果による分岐の判断の際には、この誤差に十分注意する必要があります。

整数型　ループの回数などをカウントするときに使う－32,7

	68から＋32,767までの範囲内の整数。
長整数型	大きな数のカウントや小数を含まない計算に向いた、−2,147,483,648から＋2,147,483,647までの範囲内の整数。
単精度型浮動小数点形式	一般的な計算に用いられる有効桁7桁(百万の単位)の小数点を含んだ実数。
倍精度型浮動小数点形式	高精度な計算が必要な場合に用いられる有効桁15桁(百兆の単位)の小数点を含んだ実数。

変数と型

変数は、数値や文字などのデータを格納しておく箱にたとえられます。

変数には「変数名」をつけ、必要な場合は格納するデータの型を表わす「型宣言文字」を付加します。

変数名は、40文字までの英文字（日本語文字は含まない）、数字と小数点「.」（ピリオド）を使ってつけます。

ただし、「FN」で始まる変数名は「DEF FN関数」で宣言された関数の呼び出しとみなされてしまうので、一般の変数は変数名の先頭を「FN」で始めるこ

※1バイト文字　※2バイト文字　キーボードから直接入力できるアルファベットやカナ、数字、記号と、特別な働きをする制御文字を合わせて「ASCII」(アスキー）文字といい、また1文字が1バイト（2進数で8桁）の文字コードで表わされているので、これを1バイト文字といいます。

これに対して、日本語文字は1文字が2バイトの文字コードで表わされ、アスキー文字2文字分に数えられ、文字コードでの入力以外は、日本語FEPを使わないと直接入力することができません。

Quick BASICでは、「シフトJIS」というコード体系を採用しているので、アスキー文字と漢字などの日本語文字列を1文字あたりのバイト数の違いだけで、同じように取り扱うことができます。

N88−日本語BASIC（86）などの「KI/KO」（漢字イン／漢字アウト）コード方式の場合は、このKI/KOコードを日本語文字の前後に付加するため、文字数を数えたり文字列処理をする場合、KI/KOコードの文字数を加えるなどの特別の注意が必要でしたが、Quick BASICでは、一般の文字列処理関数でも、バイト数に注意するだけで日本語文字を取り扱えるようになっています。

Quick BASICでも、日本語に関する専門の関数が多く用意されています。

とはできません。

型宣言文字は、格納するデータの型に従って、５つの型がありますが、型の宣言をしないか、型宣言文字を付加しないと、単精度型とみなされます。

型宣言文字

文字型	$	（ダラー）
整数型	%	（パーセント）
長整数型	&	（アンパーサンド）
単精度型	！	（エクスクラメーションマーク）
倍精度型	#	（シャープまたはナンバー）

型の宣言は、変数名の後ろに型宣言文字を付加しない場合には、変数を使う前に行わなければなりません。

変数の型宣言

Quick BASIC では、数多くのデータの型と、それらのデータを格納する変数の型が用意されています。

変数の型を宣言するためには「宣言ステートメント」が用意されています。

型宣言の方法は、

[宣言ステートメント] [変数名] AS [型]

のように、「変数名」に対して、「AS」節を使って「型」を割り当てます。

例えば、

```
DIM a AS STRING
```

という宣言は、変数「a」を文字型のデータを格納する変数として使用することを宣言しています。

この宣言以降、変数 a には文字型データを示す「＄」を付加する必要がなく

なります。

「宣言ステートメント」には次のようなものがあります。

SHARED	グローバル変数
STATIC	ローカル変数
DIM	配列または単純変数
REDIM	動的配列の再定義（メモリ管理のために配列を再定義する）
COMMON	共用（複数モジュールや CHAIN される他のプログラムと変数を共通に使用する）

変数の「型」の割り当てには次のようなものがあります。

INTEGER	整数型
LONG	長整数型
SINGLE	単精度浮動小数点型
DOUBLE	倍精度浮動小数点型
STRING	文字列型

このほかに、ユーザー定義型の型も宣言することができます。

文字列型は、「*」のあとに文字数を指定して、固定長文字列を宣言することもできます。

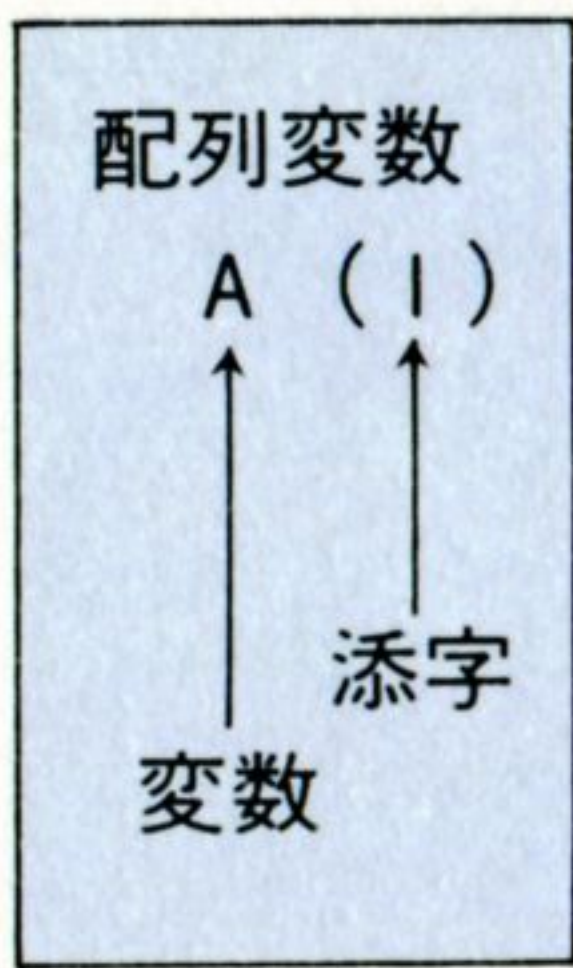
変数の型宣言は、変数のスコープを決定したり、型を事前に宣言すると同時に、どのような変数をプログラム中で使用するかの宣言にもなりますので、大きなプログラムを作成するときは、モジュールの初めで宣言をしておくようにするといいでしょう。

●配列変数

変数には、「単純変数」と「配列変数」があります。

単純変数は、単一のデータを格納しますが、配列変数は、同じ型のデータを集合体として扱い、「添字」を使って個このデータの要素を格納します。添字をいくつかの組にして、「多次元配列」を使うこともできます。

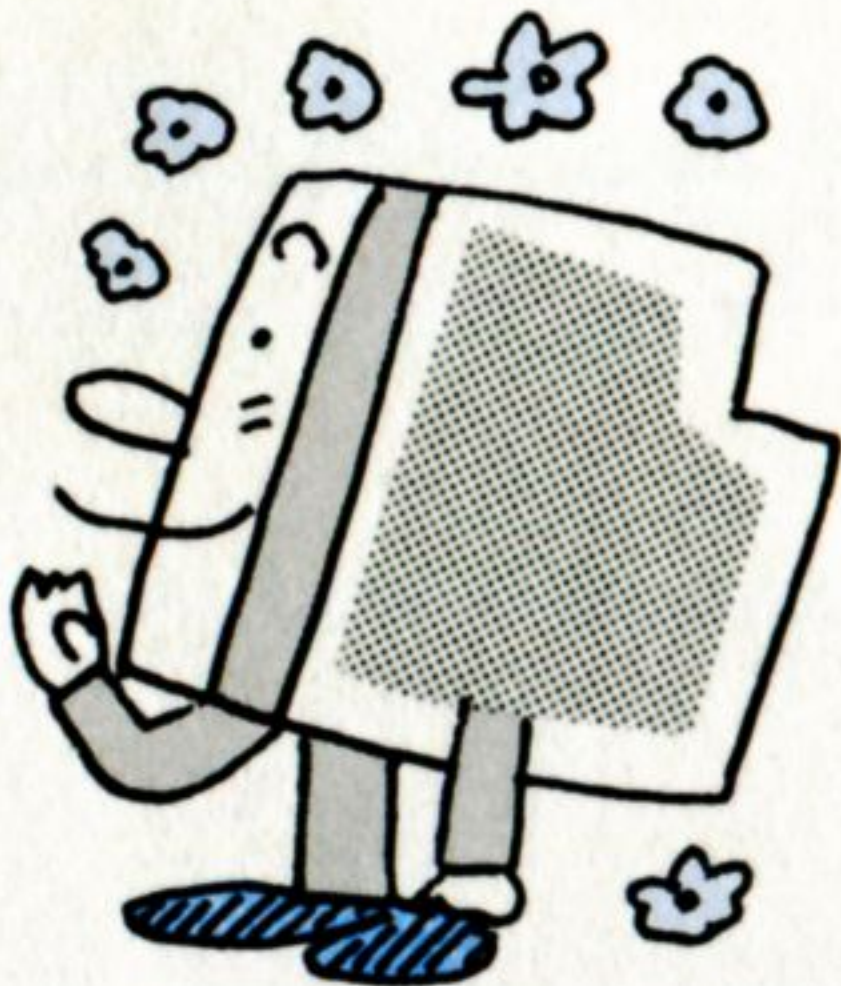
多次元配列は、六十次元まで可能ですが、一般的に二次元から三次元までを利用します。一次元配列は変数の直線的な並びに、二次元配列は変数の平面的

な並びにたとえられます。

配列変数
A（1）
↑　↑
変数　添字

一次元配列

A（1）	A（2）	A（3）	A（4）

二次元配列

A（1,1）	A（1,2）	A（1,3）
A（2,1）	A（2,2）	A（2,3）
A（3,1）	A（3,2）	A（3,3）

■配列変数と配列

配列変数も単純変数と同じように、変数名と型宣言文字を使って各種の型にすることができます。

添字は、「( )」（カッコ）でくくった整数で指定します。整数以外で指定すると、整数部分に丸められます。

```
DIM a(2Ø) AS INTEGER
```

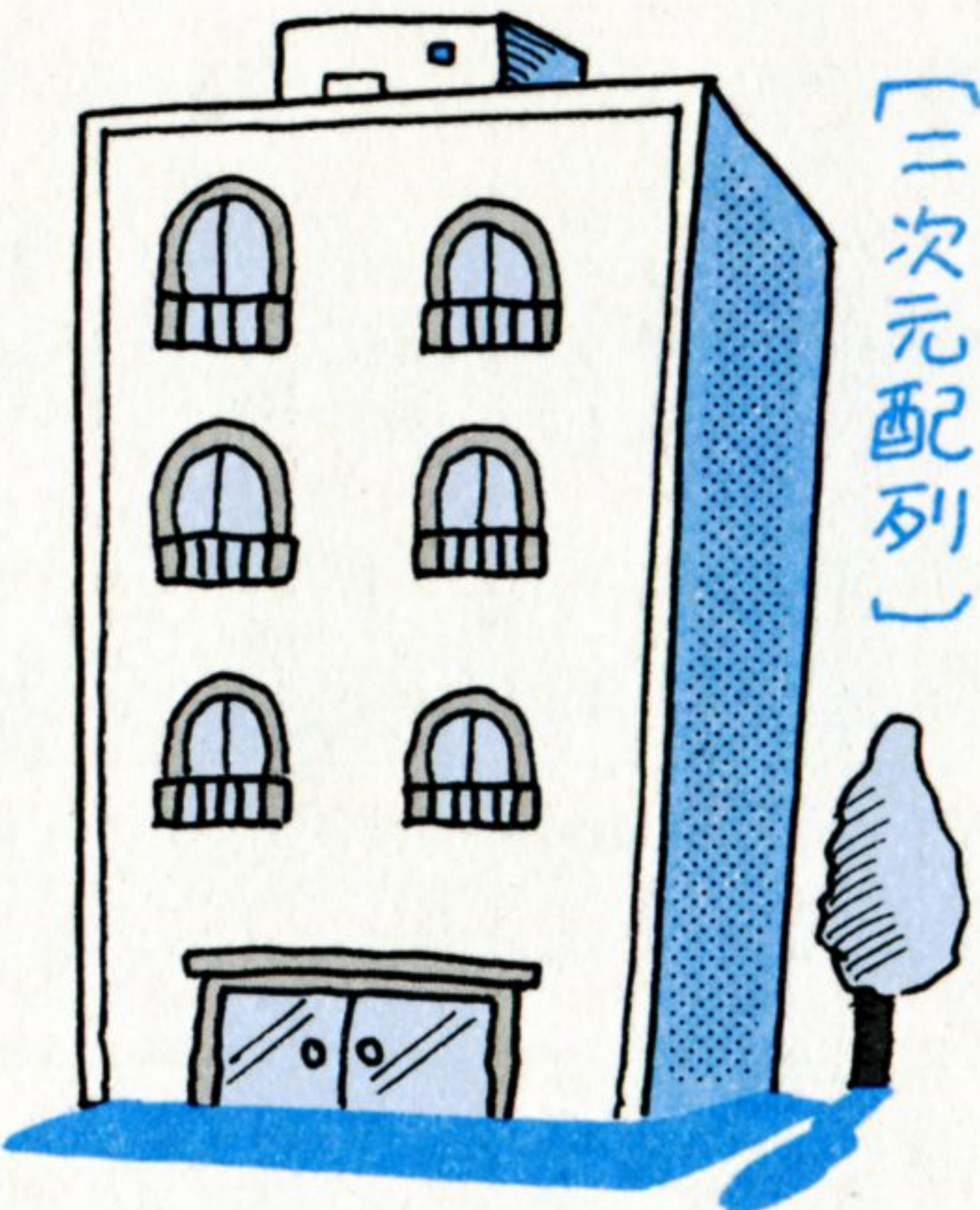
「a」という単精度の配列に20の要素を宣言しています。

添字を「，」（カンマ）で区切ってつづければ、最大六十次元までの高次元配列を宣言することができます。

一次元あたりの要素（添字）の最大は、32,768まで指定できますが、なにも指定しないときは10の要素が指定されたことになるので、特に宣言を行わなくても、最高10の添字の要素までは使うことができます。

〔一次元配列〕

〔二次元配列〕

複合データ型変数
TYPE~END TYPE
DIM

ユーザー定義変数

Quick BASICでは、用意されている変数の型を、ユーザーが組み合わせて定義すれば、「複合データ型」の変数を利用することができます。

複合データ型変数は、文字や数値の変数の各型を組み合わせて定義した新たな変数の型です。

初めに型の定義を「TYPE~END TYPE」ステートメントを使って行います。

次に変数の宣言を「DIM」ステートメントで行い、ユーザー定義変数として使えるようにします。

```
TYPE Kata
  Namae AS STRING *20
  Nenrei AS INTEGER
END TYPE
DIM Tosi AS Kata
```

20文字の文字型の要素「Namae」と、整数型の要素「Nenrei」とを組み合わせた「Kata」という複合データ型を定義して、「Tosi」というユーザー定義変数を宣言しています。

変数Tosiへのデータ代入は、要素ごとに、

```
Tosi. Namae="Mari"
Tosi. Nenrei=20
```

のように、変数名と要素名を「.」(ピリオド)で区切って指定して行います。

変数Tosiの内容は、

```
PRINT Tosi. Namae
PRINT Tosi. Nenrei
```

とすると、確認できます。

結果は、

```
Mari
20
```

で、２つのデータが格納されていることがわかります。

ユーザー定義変数では、値を個この要素ごとに代入したり表示することができますが、一括して変数の内容を表示させたり、代入することはできません。

ユーザー定義変数は、ランダムアクセスファイルの読み出し、書き込みの際に、データをひとまとめにするのに便利なので、ランダムアクセスファイルと組み合わせて使われます。

変数のスコープ

Quick BASIC では、変数をモジュール・プロシージャごとに管理しています。

ひとつの変数名は、モジュール・プロシージャ内では同じ変数として扱われますが、他のモジュールやプロシージャで同じ変数名を使っても、別の変数として扱われるため、ユーザーはプログラム全体の変数の管理を Quick BASIC にまかせておけるようになっています。

例えば、メインモジュールで、変数「ｉ」をループのカウンタとして使っていても、SUB モジュールでは、これに影響を与えずに変数「ｉ」を使うことができます。

メインモジュール

```
FOR i = 1 TO 10
    CALL KAKE
    PRINT "MAIN i="; i
NEXT
```

SUB モジュール

```
SUB KAKE
    i= i * 2
    PRINT "SUB i="; i
END SUB
```

このプログラムは、SUB モジュールで変数ｉに２をかけて、ｉに代入していますので、ｉの値が変わってしまっています。

これは、SUB モジュール「KAKE」のなかの変数ｉの値が変わっただけで、メインモジュールのループカウンタで使われている変数ｉまでは影響を与えていません。

ローカル変数
グローバル変数
スコープ　　COMMON

テストランすると、それぞれの値の変化が表示されますので、各モジュール内でしか変数に影響がないことがわかります。このように変数が影響を与える範囲のことを「スコープ」といいます。

Quick BASIC では、特に指定しないかぎり、変数のスコープが同一のプロシージャ内に限定されています。このような変数を「ローカル変数」といい、ローカルとは「局部的な」という意味で使われています。

これに対して、Quick BASIC では「グローバル変数」として特に宣言することができます。

グローバル変数は、N_{88}－日本語 BASIC（86）などの他の BASIC での変数の管理の方法と同じように、モジュール全体にわたって同じ変数名を共通して使用することができます。

```
SHARED g$ AS STRING * 10
```

と宣言すると、変数「g$」は、モジュール中のモジュールレベルコードやプロシージャで文字列を代入して内容を変えたり、内容を参照したりすることができるようになります。

プログラムの別べつのモジュール間で変数を共通に使用するためには、共有する各モジュールのモジュールレベルで、

```
COMMON u AS INTEGER
```

のように「COMMON」ステートメントを使って、変数の共用宣言します。こうすると、すべてのモジュールで変数「u」に共通に、代入・参照をすることができるようになります。

ファイルの使いこなしは

データのファイルとプログラムのファイルがあって

ファイル

「ファイル」は、フロッピーディスクなどに記録されたプログラムやデータのことです。

フロッピーディスクをオフィスのスチールキャビネットにたとえると、そのなかにしまわれる書類などをつづり込んだファイルと同じような意味です。

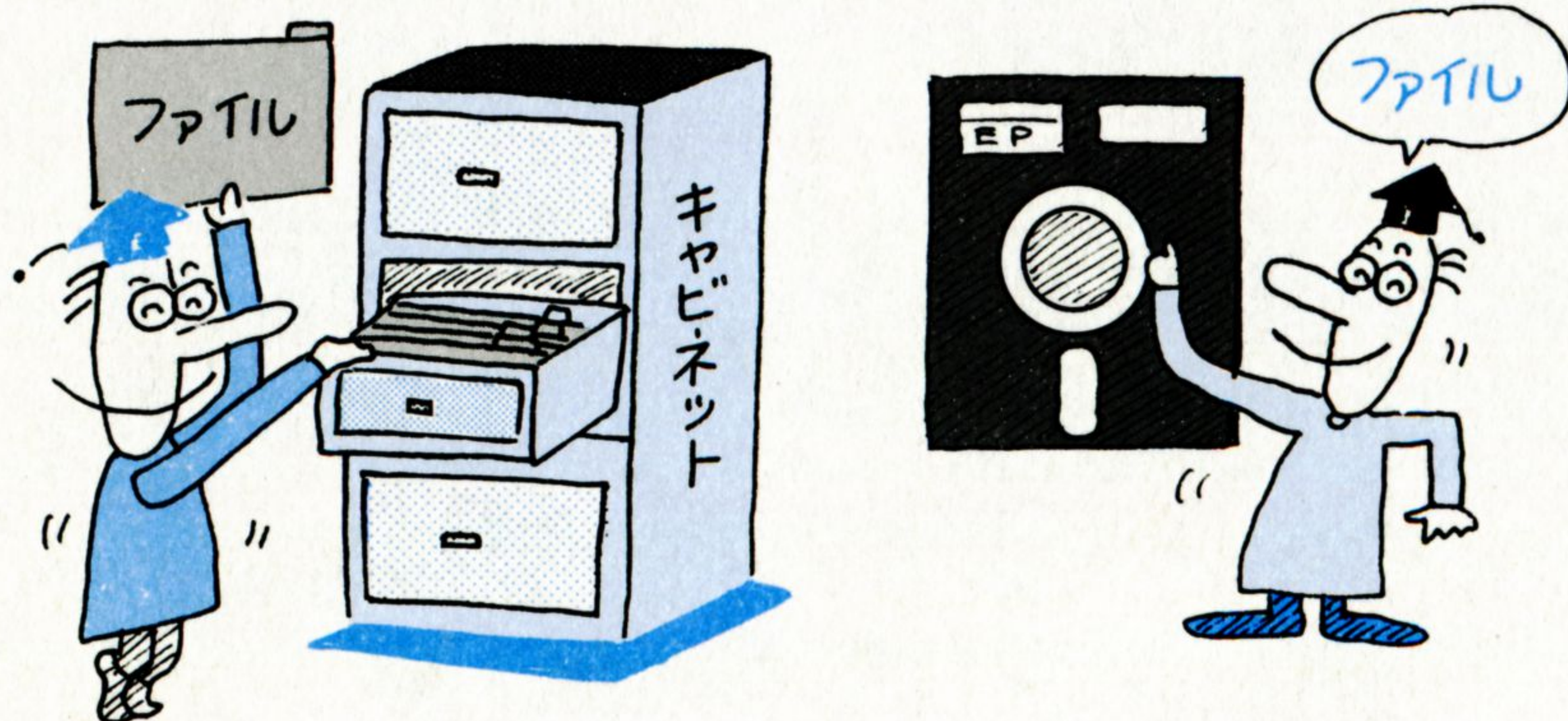
スチールキャビネットのなかのファイルには、書類や帳票がしまわれているように、フロッピーディスクのファイルのなかにも「プログラム」や「データ」が記録されています。

パソコンを使うということは、プログラムによってなにかのデータを処理することであることは、すでに触れました。

プログラムは、一度作成すれば、再度入力しなくてもよいようにフロッピーディスクなどの補助記憶装置に「プログラムファイル」として保存し、実行のときにパソコンに読み出します。

では、データはどのような形になっているのでしょうか。

ユーザーが入力する

データは、処理を必要としている人、つまりユーザーがもっていて、プログラム実行中にINPUTステートメントなどを使って、必要があるときにキーボ

ードなどから入力する方法があります。

データは、ユーザーの頭になかにあるわけです。

プログラムのなかに入れておく

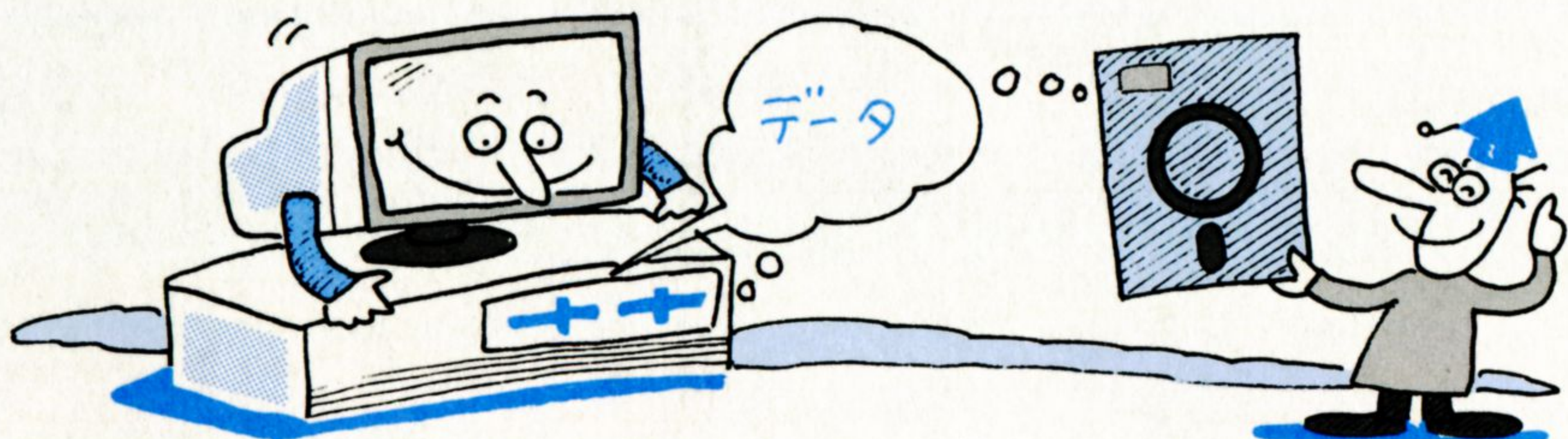
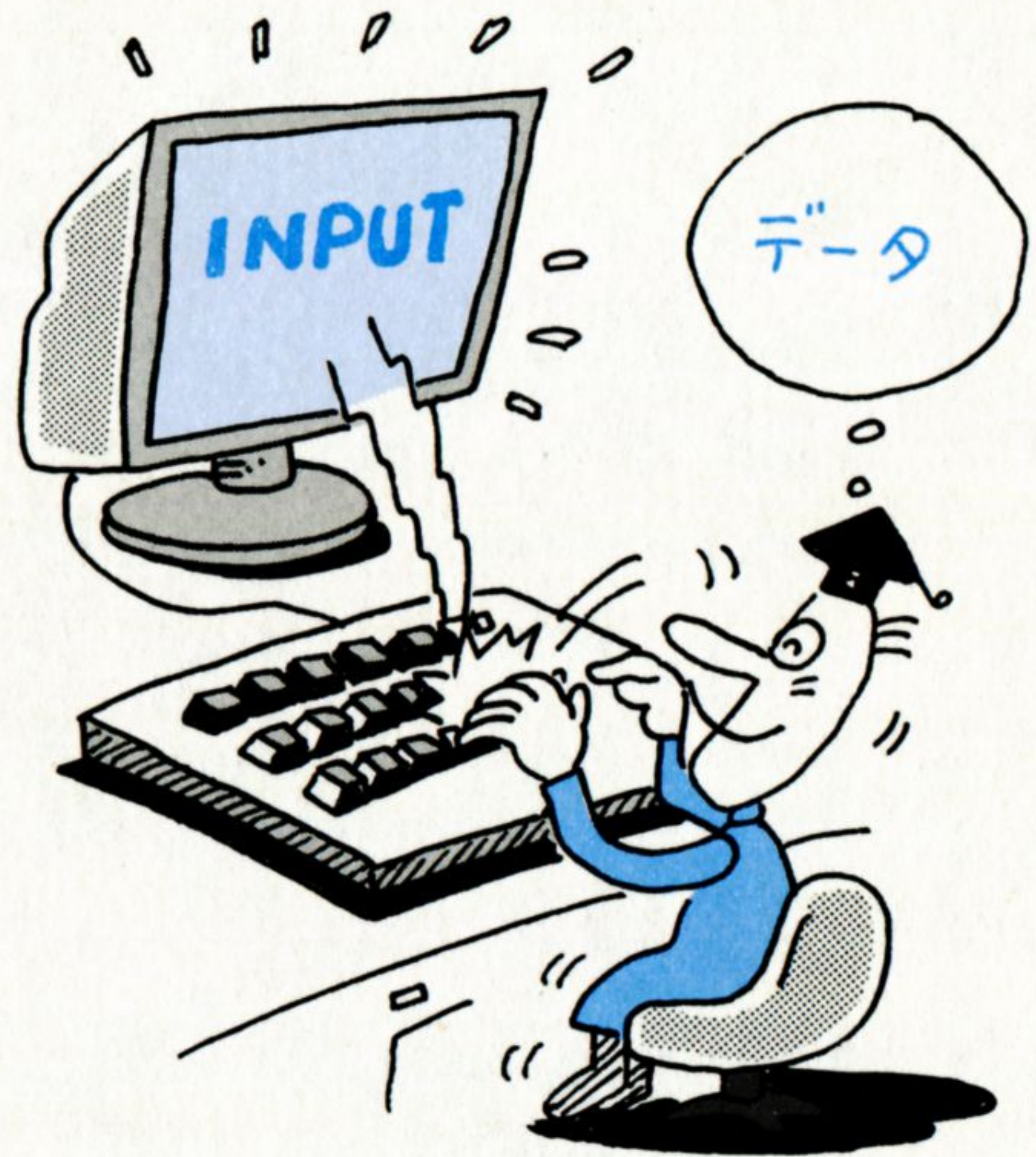
ユーザーによって変更されることのないデータではありますが、プログラマーが必要に応じて変更するようなデータは、プログラムのなかに入れておきます。

READ～DATAステートメントを利用して、データをひとまとめにしておき、変数に代入して利用するのが一般的です。

フロッピーディスクなどに保存しておく

フロッピーディスクやハードディスクなどの補助記憶装置にデータを保存しておき、必要なときに読み出して利用し、処理結果を書き込んで保存しておく方法があります。

住所録などのプログラムでは、キーボードからデータを入力して、そのデータをフロッピーディスクに記録し、表示や検索などの必要なときに、フロッピーディスクから読み出されます。

プログラムとデータを別の「ファイル」として、フロッピーディスクに保存しておけば、大量のデータを取り扱うことができます。

プログラムファイル

Quick BASICでは、プログラムをフロッピーディスクに保存する形式には、2種類あります。

● 標準ファイル形式

Quick BASIC独自のフォーマットで、テキストファイルよりも高速でプログラムを読み出すことができますが、他のエディタなどで読んだり修正したりすることはできません。

● テキストファイル形式

MS-DOS標準のASCIIファイル(テキストファイル)の形式で、他のエディタなどでも読み出しや修正ができ、またMS-DOSのTYPEコマンドで内容を表示させることもできます。

● プログラムファイルの拡張子

プログラムファイルは、拡張子で見分けることができます。

Quick BASICでは、プログラムを保存する際に、ファイルネームに拡張子を指定しないと、「.BAS」というBASICプログラムであることを示す拡張子を自動的につけてくれます。

この拡張子は、Quick BASICの環境下で、修正や実行のできるプログラムファイルであることを示しています。

また、MS-DOSのコマンドラインから直接実行できる自動実行型のプログラムファイルには、「.EXE」の拡張子を付加します。

● メイクファイル

Quick BASICでは、モジュールごとにプログラムを管理して、複数のモジュールやサブプログラム、また他のプログラムのモジュールも利用できるようになっています。

複数のモジュールからできているプログラムも、メインモジュールのファイル名で管理されていますので、ユーザーはサブプログラムの管理を行う必要がありません。

他のプログラムのモジュールを利用した場合は、メインモジュールを保存するときに、自動的に「メイクファイル」が作成されます。

メイクファイルは、参照されたモジュールのあるプログラムを、メインモジュールのサブプログラムとして読み込むために、そのプログラムに含まれるす

べてのモジュール名を記録したファイルです。

メイクファイルを利用すれば、別べつに作成したプログラムをひとつのプログラムとして実行したり、いくつかのモジュールをひとつのプログラムとして取り扱うことができるようになります。

メイクファイルは、テキスト形式のファイルで、メインモジュールと同じファイル名に「.MAK」の拡張子が自動的に付加されます。

このファイルを削除すると、サブプログラムを利用できなくなりますので、不用意に削除や修正を行わないようにしましょう。

●インクルードファイル

Quick BASICのプログラムを直接実行型のプログラムにコンパイルするとき、「インクルードファイル」を用いる場合があります。

インクルードファイルは、マルチモジュールプログラムを作成して、プログラム全体や、一部の関連するプログラムに必要な宣言ステートメントなどをまとめておくためのファイルです。

一般によく用いられる宣言ステートメントには、

DECLARE	SUB・FUNCTION プロシージャの使用宣言
DIM	変数の型宣言
TYPE	ユーザー定義型変数宣言
COMMON	複数モジュールでの変数の共用宣言

などがあります。

特に大きなプログラムをコンパイルしたり、グループでプログラムを開発した場合など、他のプログラムで多用する宣言をまとめておき、「$INCLUDE

メタコマンド」で、一定の宣言を複数のプログラムで共用する場合に使用します。

$INCLUDE メタコマンドは、一般にプログラムの先頭に置いて、コンパイル時に取り込むインクルードファイルを指定する命令です。ただし、普通のステートメントやコマンドと違って、REM ステートメントにつづけて指定します。

'$INCLUDE' [インクルードファイル名] '

「'」（クォーテーション）に囲まれた「インクルードファイル名」のファイルがコンパイル時に参照されて、本体プログラムといっしょにコンパイルされます。

インクルードファイルの作成と編集は、Quick BASIC のエディタで行うことができ、必ずテキスト形式のファイルで保存されます。一般に「.BI」の拡張子を付加します。

データファイル

フロッピーディスクへのデータの記録方法は、「シーケンシャルファイル」と「ランダムアクセスファイル」の2種類のデータファイル形式があります。

●シーケンシャルファイル

長さの決まっていないデータは、「区切り記号」を使って、1データを1フィールドとして直線的に記録されます。これがシーケンシャルファイルです。

区切りには、「,」（カンマ）を使い、データの最後には「CR−LF」（キャリッジリターンーラインフィード）記号が付加されます。

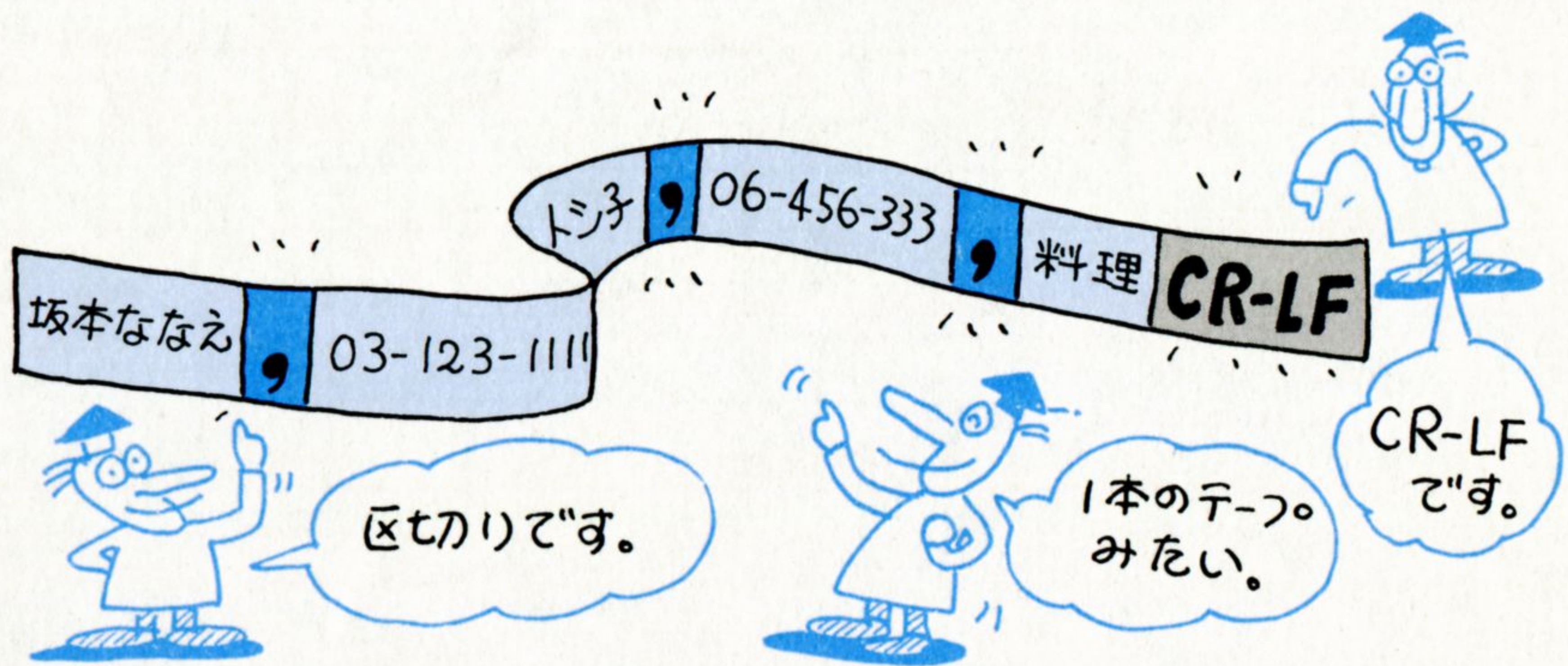
例えば、次のようなデータをシーケンシャルファイルにして記録する場合を考えてみましょう。

坂本ななえ	03-123-1111	バイク
佐藤恵子	0222-23-2222	ジョギング
三浦トシ子	06-456-3333	料理

このデータは、名まえ、電話番号、趣味の3つのデータをひとかたまりとしています。

シーケンシャルファイルでは、このデータを区切りの記号を使って記録します。イメージとしては、右ページ上の絵のような感じです。

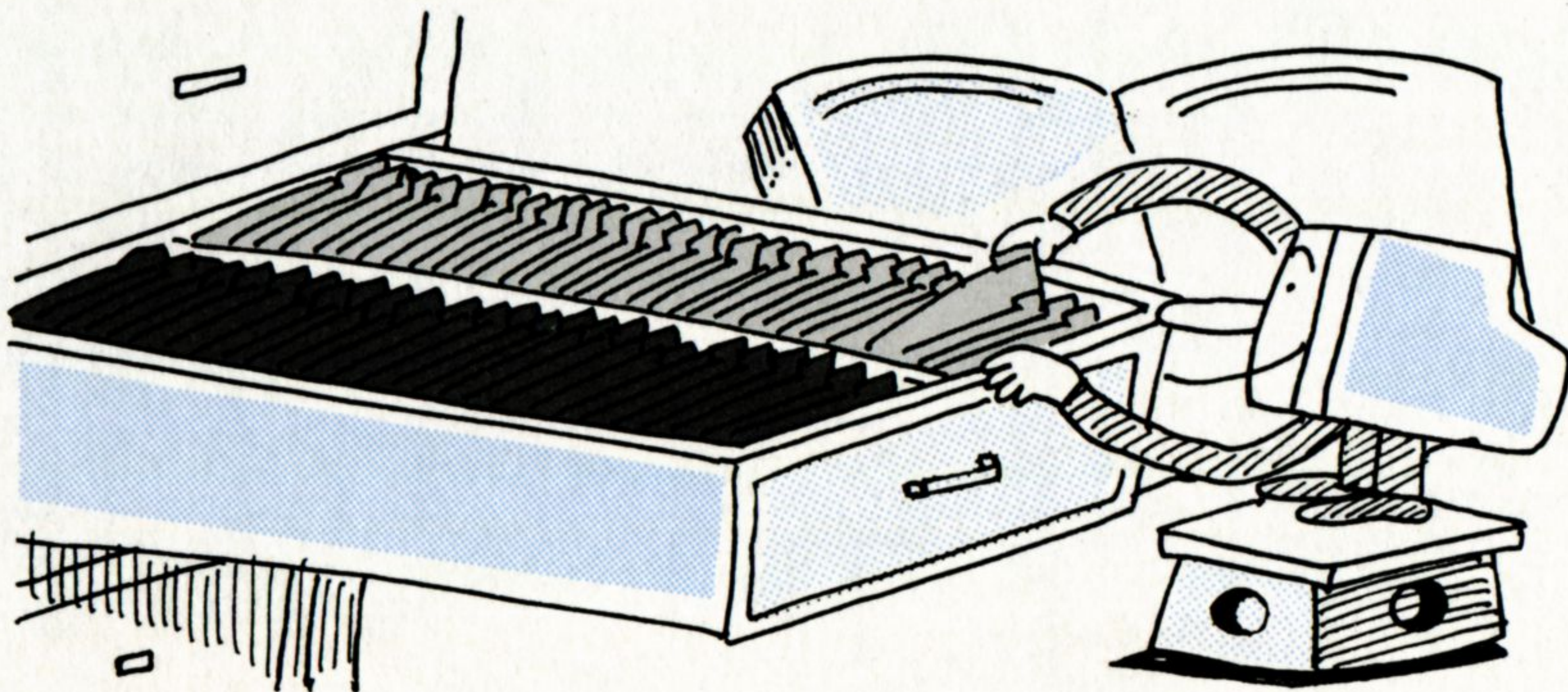
●ランダムアクセスファイル

データの長さをあらかじめ決めておいて、この長さごとにデータを区切って、平面的に配置していきます。

■シーケンシャルファイルのイメージ

先ほどのデータをランダムアクセスファイルにして記録すると、下の表のように記録され、絵のように任意のデータを指定して取り出せます。

ランダムアクセスファイルでは、データが横に名まえ、電話番号、趣味と並び、縦に１人めから３人めまで、きちんと並んでいるのがわかります。

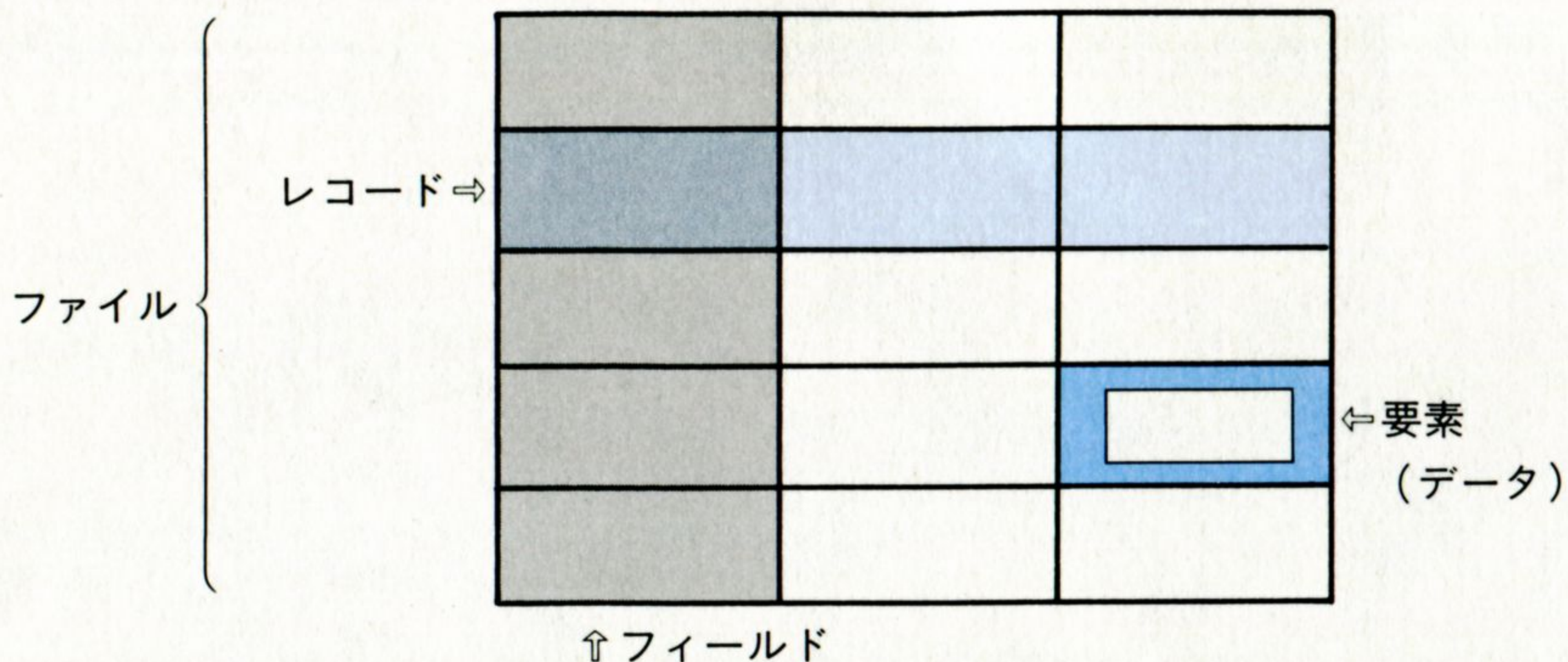
横１列の１人分のデータを「レコード」といい、レコードを構成する名まえ、電話番号、趣味の３つの要素を「フィールド」といいます。

坂本ななえ	03-123-1111	バイク
佐藤恵子	0222-23-2222	ジョギング
三浦トシ子	06-456-3333	料理

■ランダムアクセスファイルのイメージ

パソコンを利用するとき、フィールド、レコード、ファイルの概念は、たいせつなので、下図に示します。

レコード⇨

ファイル

⇦要素（データ）

⇧フィールド

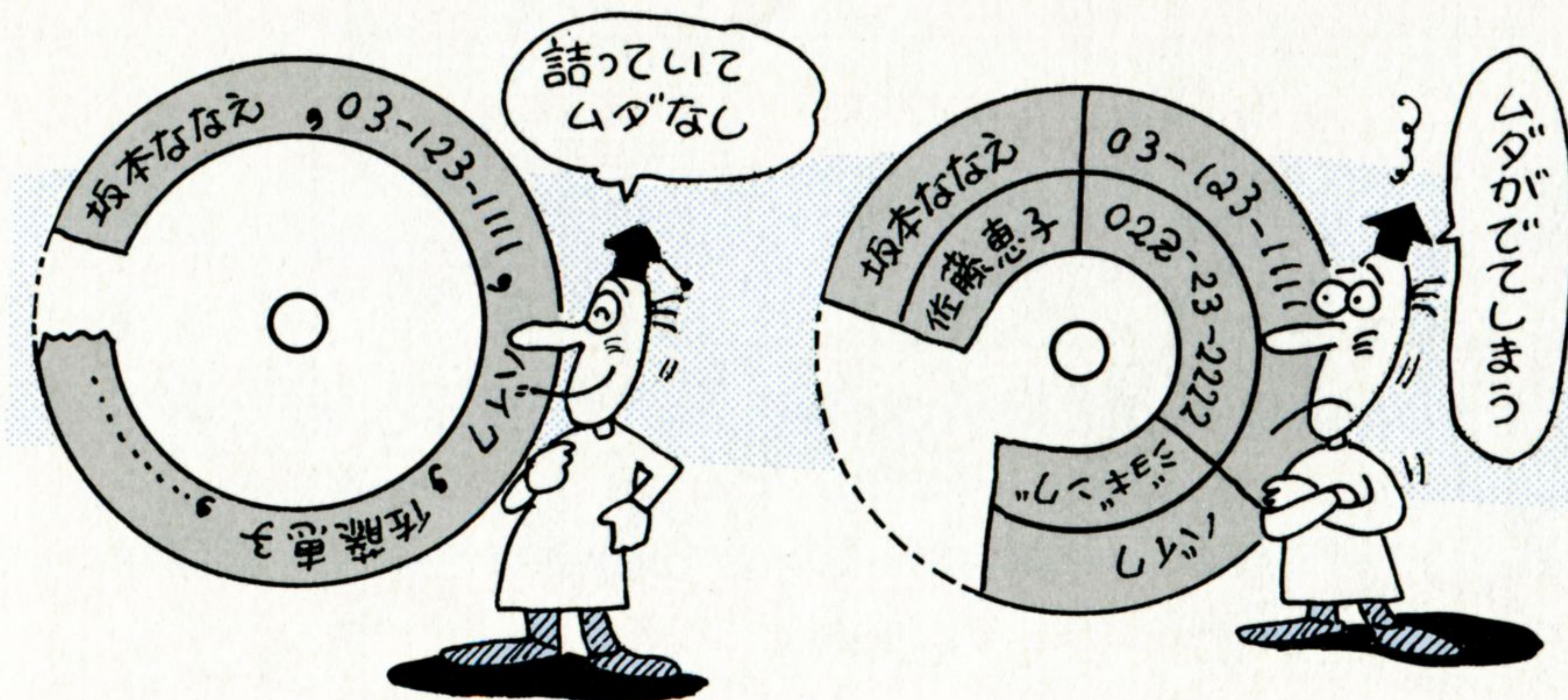
■フィールド、レコード、ファイルの概念

●シーケンシャルファイルとランダムアクセスファイルの比較

シーケンシャルファイル、ランダムアクセスファイルのそれぞれの長所短所をくらべてみましょう。

シーケンシャルファイルは、データの長さを気にすることなく、次つぎとデータを記録していくことができます。

データとデータの間に、区切りの記号を置くだけで、連続的にデータを記録していくので、フロッピーディスクにむだが出ません。

これに対して、ランダムアクセスファイルは、あらかじめひとつひとつのフィールドの長さを決めてしまうので、データの長さがフィールドに割り当てら

シーケンシャルファイル　　　ランダムアクセスファイル

れた長さより短くても、あまりをスペースで埋めてしまいます。

逆に、データがフィールドより長いと、データの一部が切り取られてしまいます（このようなことが起こらないように、フィールド長はデータの長さを予想して、十分な長さを確保しておきます）。

シーケンシャルファイルは、１ファイルの先頭がどこにあるのかをパソコンが管理しています。

これに対して、ランダムアクセスファイルをフロッピーディスクに作成すると、パソコンは１レコードずつ、フロッピーディスクのどこにデータを記録したかを管理しています。

このため、シーケンシャルファイルは、データを読み出したり書き込んだりするときに、１ファイルごとに、最初から最後までを順番に、すべてのデータを読み出し、書き込まなければなりません。

ランダムアクセスファイルは、データがレコードごとにきちんと管理されているので、何番目のレコードを読むと指定すれば、そのレコードの部分だけを読み出したり、書き込んだりすることができます。

これは、ちょうどカセットテープとCDで、途中に録音されている曲を探し出すことにたとえられます。シーケンシャルファイルは、カセットテープのように、初めの曲から順番に探していきますが、ランダムアクセスファイルはCDのように、特定の曲をボタンひとつで聞き始めることができるのと同じです。

フロッピーディスクにデータを記録する場合、どちらの方法がいいか、データの種類や処理の方法によって適切な方法を選ぶ必要があります。

シーケンシャルファイルに向いたデータ

データの長さが決まっていない
データの件数が少ない
全データをオンメモリ（一括して読み出す）で処理する

ランダムアクセスファイルに向いたデータ

データの形式があらかじめ決まっている
データの件数が多い
全データをオンメモリで処理しない

配列とシーケンシャルファイル

シーケンシャルファイルには、データが一定の順番で直線的に記録されているので、データを初めから終わりまで一括してフロッピーディスクから読み出して、処理するのに向いています。

シーケンシャルファイルのデータを読み出したり、書き込んだりするのに、配列変数とループを使います。

ループは一定の作業を繰り返すので、順番に並んでいるデータを読み出したり、書き込んだりするのに好都合です。

配列変数は、添字の数値をループカウンタに合わせて変えていくと、一定の規則でデータを読み出して、配列変数に格納していくことができます。

配列変数に格納されたデータは、オンメモリのデータですから、データの処理を行う際に、フロッピーディスクにアクセスする必要がなく、高速処理をすることができますが、メモリの量によっては、処理可能なデータの量が制限されてしまいます。

配列変数を使ってシーケンシャルファイルからデータを格納するところをイメージしてみると、こんな絵のような感じでしょう。

ファイルを作成するプログラムと、フロッピーディスクのファイルからデータを読み出すプログラムを実際につくってみます。

ファイルを取り扱うには、まず読み書きするファイルを指定して「オープン」しなければなりません。

ファイルのオープンは、ちょうどファイリングキャビネットから必要なファ

イルを一冊取り出して、そのファイルを開く作業にたとえられ、「OPEN」ステートメントを使って、ファイルを使用するための各種の指定を行います。

ファイルをオープンして、データファイルの利用を可能にしてから、ループと配列変数を使ってファイルを読み出したり、書き込む処理を行います。

● シーケンシャルファイルを書き込むプログラム

※左側の番号と：(コロン)は入力しないでください

```
 1: '****************************************************
 2: '            シーケンシャルファイル書き込み
 3: '****************************************************
 4:
 5: DIM Namae$(10), Denwa$(10), Shumi$(10)
 6:
 7: FOR i = 1 TO 3
 8:     READ Namae$(i), Denwa$(i), Shumi$(i)
 9: NEXT
10: DATA 坂本ななえ,03-123-1111,バイク
11: DATA 佐藤恵子,0222-23-2222,ジョギング
12: DATA 三浦トシ子,06-456-3333,料理
13:
14:
15: OPEN "TECHO.DAT" FOR OUTPUT AS #1
16:
17: FOR i = 1 TO 3
18:     WRITE #1, Namae$(i), Denwa$(i), Shumi$(i)
19: NEXT
20:
21: CLOSE #1
22: END
23:
```

● シーケンシャルファイルを読み出すプログラム

```
 1: '****************************************************
 2: '            シーケンシャルファイル読み出し
 3: '****************************************************
 4:
```

```
 5: DIM Namae$(10), Denwa$(10), Shumi$(10)
 6:
 7:
 8: OPEN "TECHO.DAT" FOR INPUT AS #1
 9:
10: i = 0
11: DO UNTIL EOF(1)
12:     i = i + 1
13:     INPUT #1, Namae$(i), Denwa$(i), Shumi$(i)
14: LOOP
15:
16: CLOSE #1
17:
18: FOR j = 1 TO i
19:     PRINT Namae$(j), Denwa$(j), Shumi$(j)
20: NEXT
21:
22: END
23:
```

ユーザー定義変数とランダムアクセスファイル

ランダムアクセスファイルは、まず初めにデータの形を十分検討して、各フィールドの長さや、データの型、1レコードの構成などを決定しておかなければなりません。

ランダムアクセスファイルは、データが1レコードずつ一定の形式で記録されていますから、データのなかから特定の1レコードを直接に読み出して、処理するのに向いています。

ランダムアクセスファイルのデータは、ユーザー定義型の変数を利用して、1レコードのデータを一括して読み出したり、書き込んだりします。

読み出すレコードの指定には、「レコード番号」を使います。

データは、ランダムアクセスファイルにレコード番号を指定して書き込み、読み出すときは、そのレコード番号を指定します。

このレコード番号を管理するために、普通「インデックスファイル」というシーケンシャルファイルを使います。インデックスファイルは、データの見出しとなるキーデータとレコード番号を結びつける索引のファイルです。

レコード番号
キーデータ
インデックスファイル

ランダムアクセスファイルを使用する前に、キーデータとレコード番号をインデックスファイルから、配列変数に読み出しておき、この配列変数のデータをもとに、ランダムアクセスファイルのデータを読み出したり、書き込んだりします。

例えば、キーデータとして「名まえ」を使っている場合、インデックスファイルを使ってレコード番号を管理すると、

1．インデックスファイルからキーデータとレコード番号のデータを読み出しておく

2．キーデータのなかから詳しいデータを取り出したい人の名まえを見つける

3．その名まえのレコード番号のデータを、ランダムアクセスファイルから読み出してくる

という手順になります。

実際にランダムアクセスファイルに書き込むプログラムと、フロッピーディスクのファイルからデータを読み出すプログラムをつくってみましょう。

ランダムアクセスファイルにデータを書き込むときには、インデックスファイルにもレコード番号を書き込んでおく必要がありますから、2つのデータファイルを同時にオープンします。

インデックスファイルとランダムアクセスファイルを使って、データを格納するようすを確認してみてください。

●ランダムアクセスファイルに書き込むプログラム

※左側の番号と：(コロン)は入力しないでください

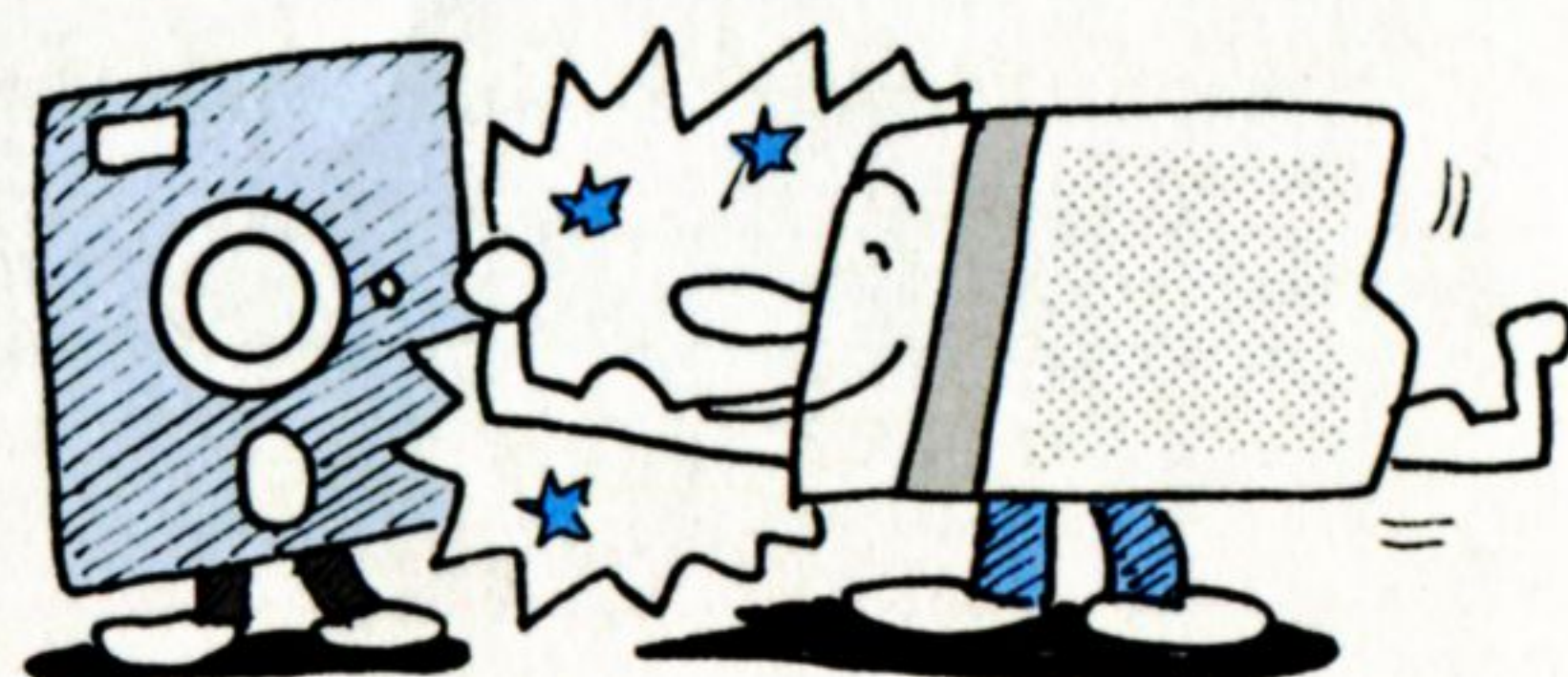
```
1: '**************************************************
2: '           ランダムアクセスファイル書き込み
3: '**************************************************
4:
5: DIM Namae$(10), Denwa$(10), Shumi$(10)
6:
7: FOR i = 1 TO 3
8:     READ Namae$(i), Denwa$(i), Shumi$(i)
9: NEXT
```

```
10: DATA 坂本ななえ,03-123-1111,バイク
11: DATA 佐藤恵子,0222-23-2222,ジョギング
12: DATA 三浦トシ子,06-456-3333,料理
13:
14: TYPE Kata
15:     n AS STRING * 20
16:     d AS STRING * 12
17:     s AS STRING * 30
18: END TYPE
19:
20: DIM Techo AS Kata
21:
22:
23: OPEN "TECHO.RDT" FOR RANDOM AS #1 LEN = LEN(Techo)
24:
25: FOR RecNumber = 1 TO 3
26:     Techo.n = Namae$(RecNumber)
27:     Techo.d = Denwa$(RecNumber)
28:     Techo.s = Shumi$(RecNumber)
29:
30:     PUT #1, RecNumber, Techo
31:
32: NEXT
33:
34: CLOSE #1
35: END
```

●ランダムアクセスファイルに書き込むプログラム（インデックス付き）

```
 1: '**************************************************
 2: ' ランダムアクセスファイル書き込み (インデックス)
 3: '**************************************************
 4:
 5: max = 100
 6: DIM IndexKey$(max), IndexRec(max)
 7:
 8: TYPE Kata
 9:     n AS STRING * 20
10:     d AS STRING * 12
```

※左側の番号と：(コロン)は入力しないでください

```
11:     s AS STRING * 30
12: END TYPE
13:
14: DIM Techo AS Kata
15:
16:
17: OPEN "TECHO.RDT" FOR RANDOM AS #1 LEN = LEN(Techo)
18: i = 0
19: DO
20:     i = i + 1
21:     RecNumber = i
22:     INPUT "名前の読み : "; NamaeKey$
23:     IndexKey$(i) = NamaeKey$
24:     IndexRec(i) = RecNumber
25:
26:     INPUT "名前        : "; Techo.n
27:     INPUT "電話        : "; Techo.d
28:     INPUT "趣味        : "; Techo.s
29:     PRINT
30:     PUT #1, RecNumber, Techo
31:
32: LOOP WHILE NamaeKey$ <> ""
33: max = i - 1
34: CLOSE #1
35:
36: OPEN "TECHO.IND" FOR OUTPUT AS #1
37:
38: FOR i = 1 TO max
39:     WRITE #1, IndexKey$(i), IndexRec(i)
40: NEXT
41:
42: CLOSE
43: END
```

●ランダムアクセスファイルから読み出すプログラム

```
 1: '**************************************************
 2: '           ランダムアクセスファイル読み出し
 3: '**************************************************
 4:
 5: TYPE Kata
 6:     n AS STRING * 20
 7:     d AS STRING * 12
 8:     s AS STRING * 30
 9: END TYPE
10:
11: DIM Techo AS Kata
12:
13:
14: OPEN "TECHO.RDT" FOR RANDOM AS #1 LEN = LEN(Techo)
15:
16: RecSuu = LOF(1) ¥ LEN(Techo)
17:
18: FOR RecNumber = 1 TO RecSuu
19:
20:     GET #1, RecNumber, Techo
21:
22:     PRINT "名前 : "; Techo.n
23:     PRINT "電話 : "; Techo.d
24:     PRINT "趣味 : "; Techo.s
25:     PRINT
26:
27: NEXT
28:
29: CLOSE #1
30: END
```

※左側の番号と：(コロン)
は入力しないでください

●ランダムアクセスファイルから読み出すプログラム（インデックス付き）

```
 1: '**************************************************
 2: ' ランダムアクセスファイル読み出し（インデックス）
 3: '**************************************************
 4:
 5: max = 100
```

※左側の番号と：(コロン)
は入力しないでください

```
 6: DIM IndexKey$(max), IndexRec(max)
 7:
 8: OPEN "TECHO.IND" FOR INPUT AS #1
 9: i = Ø
1Ø: DO UNTIL EOF(1)
11:     i = i + 1
12:     INPUT #1, IndexKey$(i), IndexRec(i)
13: LOOP
14: MaxRec = i
15:
16: TYPE Kata
17:     n AS STRING * 2Ø
18:     d AS STRING * 12
19:     s AS STRING * 3Ø
2Ø: END TYPE
21:
22: DIM Techo AS Kata
23:
24:
25: OPEN "TECHO.RDT" FOR RANDOM AS #2 LEN = LEN(Techo)
26:
27: DO
28:     INPUT "探す人の名前 : "; NamaeKey$
29:     IF NamaeKey$ = "" THEN EXIT DO
3Ø:     FOR i = 1 TO MaxRec
31:         IF INSTR(IndexKey$(i), NamaeKey$) <> Ø THEN
32:
33:             GET #2, IndexRec(i), Techo
34:
35:             PRINT "名前 : "; Techo.n
36:             PRINT "電話 : "; Techo.d
37:             PRINT "趣味 : "; Techo.s
38:             PRINT
39:         END IF
4Ø:     NEXT
41: LOOP WHILE NamaeKey$ <> ""
42:
43: CLOSE
44: END
```

★ Quick BASIC 便利メモ★

――― N_{88}-日本語 BASIC(86)から Quick BASIC へ移植するときの注意 ―――

NEC製のN_{88}-日本語BASIC（86）のプログラムをQuick BASICに移植する場合、Quick BASICのヘルプメニューにも一部案内がありますが、そのほかの特に注意する点をいくつかあげておきます。

■DOSの違い■

NEC製の標準BASICは、MS-DOS版のN_{88}-日本語BASIC（86）のファイル変換ユーティリティを使って、MS-DOSのテキスト（ASCII）形式ファイルに変換して、Quick BASICで読み込んでおかなければなりません。

■ステートメントの違い■

実行制御ステートメント

・IF～THEN の飛び先に直接行番号を指定することはできない。GOTO を使って指定する。

・ラベル名は、前の「*」を取って、後ろに「：」をつける。

テキスト画面表示

・CONSOLE は、Quick BASIC の VIEW PRINT に変更する。

・LOCATE は、行、列の順で指定するので、（X，Y）を逆にする。

・COLOR の色指定のパレット番号が違う。番号をつけ直すか変数にする。

グラフィック画面表示

・SCREEN の指定が違う。モードに PC-9801 のスーパーインポーズをさす「0」を指定する。

・WINDOW には SCREEN オプションを設定しないと、表示の上下が逆転する。

・CLS でクリアする画面の指定が違う。全画面のクリアは「0」を指定。

・CIRCLE での塗りつぶしを指定することはできない。

その他

・DEFtype での範囲の指定に使う区切り記号は、「－」に変更する。

・DATE$の年は西暦4桁で返される。設定は2桁でも受け付けるので、年を使った計算や文字列操作の場合、注意する。

PART 5

Quick BASIC プログラマー

パソコンと対話する

ここでは、実際にQuick BASICを使ってプログラムをつくりながらステートメントの使い方を覚えます。ここで取り上げるステートメントは、どれも基本的なものばかりです。サンプルプログラムには、実用性や楽しさを考慮したものを選びましたので、参考にして自分に合ったものに改良していくといいでしょう。

ごあいさつプログラム　PRINTを使って

PRINT（プリント）ステートメント

書式　PRINT　表示データ　区切り記号　表示データ　………

機能　指定されたデータをディスプレイに表示する

```
 1: '************************************************
 2: '                  ごあいさつプログラム
 3: '************************************************
 4: '
 5: 'goaisatu.bas
 6: '
 7:     PRINT "こんにちは　私は　Quick　BASIC　です"
 8:     PRINT
 9:     PRINT "あなたの　お名前は　なんとおっしゃいますか?"
10:     PRINT
11:
12:
13: '名前をたずねます
14:     INPUT "キーボードから打ち込んでください"; name$
15:
16:     PRINT
17:     PRINT
18:     PRINT
19:
20:
21: 'ごあいさつします
22:     PRINT name$; "　さん　こんにちは!"
23:     PRINT
24:     PRINT "これからも　よろしくお願いしまーす!!"
25:
26: END
```

※左側の番号と：（コロン）は入力しないでください

PRINTステートメントは、ディスプレイに文字や変数の内容などを表示するステートメントです。

PRINTの次にスペースをあけ、表示データを指定すると、ディスプレイにそのデータが表示されます。

Quick BASICでは、行番号を使いません。ここでは説明のために、プログラムリストを打ち出す際に、各行の先頭に行を示す番号を特につけ加えています。

すべてのサンプルプログラムにこの番号をつけ加えていますので、プログラ

ムを入力する場合は、各行の先頭の数字と「：」（コロン）、１文字分のスペースは入力しないでください。

■プログラムの使い方

プログラムを実行するとメッセージが表示され、ユーザーの名まえをキーボードから入力するように求めてきます。

ユーザーの名まえを入力して、⏎キーを押すと、Quick BASICからのごあいさつが表示されます。

■プログラムの要点

1～3行　このプログラムの名まえを示すREMステートメントです。大きなプログラムの場合は、機器などのシステム構成や作者名、作成年月日、改良年月日なども示すといいでしょう。

5行　プログラムのファイル名を示しています。

7～10行　7行からがこのプログラムの本体です。PRINTステートメントのあとにメッセージが指定されています。

14行　name$変数に、ユーザーの名まえをキーボードから入力するように求めています。

22行　PRINTにつづけて、変数name$の内容を表示しています。「；」（セミコロン）を区切りにして「　さん　こんにちは！」という文字列を表示します。

理想体重プログラム　INPUTを使って

INPUT（インプット）ステートメント

書式　INPUT　”プロンプト文”　区切り記号　変数
機能　キーボードからデータを入力して変数に格納する

INPUTステートメントは、プログラムのなかで、キーボードを使って直接ユーザーからパソコンにデータを与える方法のひとつです。キーボードから入力されたデータは、INPUTステートメントの後ろに指定された変数に格納されます。

■プログラムの使い方

プログラムを実行すると、プログラムの説明が表示され、身長の入力が求められるので、理想体重を知りたい人の身長をcm単位でキーボードから入力します。次に体重の入力を求められるので、同じようにkg単位で入力します。

```
 1: '**************************************************
 2: '             ローレル指数と理想体重プログラム
 3: '**************************************************
 4:
 5: 'taijuu.bas
 6:
 7:     PRINT "ちゃんと運動してますか?  私が理想体重を教えてあげましょう"
 8:     PRINT "あなたの  身長  と  体重  を教えてください"
 9:     PRINT
10:     PRINT
11:     INPUT "身長(cm)  は  "; sinchou
12:     PRINT
13:     INPUT "体重(Kg)  は  "; taijuu
14:     PRINT
15:
16: 'ローレル指数を計算します
17:     rohrer = INT((taijuu / sinchou ^ 3 * 10 ^ 7) * 10) / 10
18:
19:
20: '理想体重を計算します
21:     risou.ue = INT((sinchou ^ 3 * 144 / 10 ^ 7) * 10) / 10
22:     risou.sita = INT((sinchou ^ 3 * 115 / 10 ^ 7) * 10) / 10
23:
24:
25: '発表します
26:     PRINT
27:     PRINT
28:     PRINT "あなたのローレル指数は  "; rohrer; "  です"
29:     PRINT
30:     PRINT "              ～   ９９      やせすぎ"
31:     PRINT "      １００  ～  １１４     やせぎみ"
32:     PRINT "      １１５  ～  １４４     標準"
33:     PRINT "      １４５  ～  １５９     ふとりぎみ"
34:     PRINT "      １６０  ～             ふとりすぎ"
35:     PRINT
36:     PRINT "標準に入りましたか?"
37:     PRINT "それでは  あなたの理想的な体重をお知らせしましょう"
38:     PRINT
39:     PRINT "理想体重の上限  "; risou.ue; " Kg"
40:     PRINT "          下限  "; risou.sita; " Kg"
41:     PRINT
42:     PRINT "理想体重からはずれた方  運動や食事に注意しましょう"
43:
44: END
```

※左側の番号と：（コロン）は入力しないでください

すぐにローレル指数を計算し、結果と指数の目安、理想体重の上限、下限を表示して終了します。つづけて他の理想体重を知りたいなら再度実行します。

■プログラムの要点

11行　INPUT ステートメントに適切なプロンプト文を使って身長のデータ入力を求めています。

12行　次のデータ入力との間に1行間隔をあけるために、PRINTステートメントを使って1行改行しています。

17行　ローレル指数の計算を行っています。変数は、どんなデータかがわかるように、taijuuとかsinchouのような具体的な変数名をつけます。

28行　コメントで囲んで、計算結果のローレル指数を表示します。

30～34　ローレル指数の値の意味を示します。

36～42　理想体重を示して、注意事項を案内しています。

パソコンにしごとさせる

数字当てゲーム　IF～THEN～ELSEを使って

パソコンの出す問題をあなたがカンを頼りに当てていこうというゲームです。

IF～THEN～ELSE（イフ～ゼン～エルス）ステートメント

書式　IF　条件　THEN
　　　　実行ステートメント1
　　ELSE
　　　　実行ステートメント2

機能　条件によってプログラムの実行を制御する

IF～THENステートメントは、与えられた条件を判断して、その結果が成立すれば、THEN以下の実行ステートメント1を実行し、成立しなければELSE以下の実行ステートメント2を実行します。

※左側の番号と：（コロン）は入力しないでください

```
1: '**************************************************
2: '                    数字あてゲーム
3: '**************************************************
4: '
5: 'suujiate.bas
6: '
7: hajime:                  '----ゲームのはじめ
8:     CLS
9:     PRINT "私が　1　から　100　までのある数字を考えます"
10:    PRINT "その数字を　あなたが当てるゲームをしましょう"
11:    PRINT
12:    PRINT "私は　あなたの予想した数が　私の考えた数より"
13:    PRINT "大きいか　小さいか　だけを答えますので　よーく考えてください"
14:    PRINT
```

```
15:
16: mondai.tukuri:              '----問題を作ります
17:     RANDOMIZE TIMER
18:     mondai = INT(RND * 100 + 1)
19:
20:     PRINT "数字が決まりました"
21:     PRINT "それでは　はじめましょう！！"
22:     PRINT
23:
24: handan:                     '----回答の数字を判断します
25:     kai = 1
26:
27: kurikaesi:                  '----判断を繰り返します
28:     PRINT
29:     PRINT kai; "　回め　";
30:     INPUT "さあ　いくつでしょうか　"; kaitou
31:     kai = kai + 1
32:
33:     IF mondai < kaitou THEN
34:         COLOR 6, 0
35:         PRINT "大きいですね"
36:         COLOR 7, 0
37:     ELSEIF kaitou < mondai THEN
38:         COLOR 2, 0
39:         PRINT "小さいですね"
40:         COLOR 7, 0
41:     END IF
42:
43:     IF mondai = kaitou THEN GOTO seikai:  ELSE GOTO kurikaesi:
44:
45: seikai:                     '----正解です
46:     COLOR 3, 0
47:     PRINT "そのとおり　正解です　やったね！！"
48:     PRINT
49:     COLOR 7, 0
50:
51: mouitido:                   '----もう一度やるか確認します
52:     PRINT "もう一度やりましょう"
53:     PRINT
54:     PRINT "もう一度やるなら「ｙ」　やめるなら「ｎ」を押して";
55:     INPUT kakunin$
56:
57:     IF kakunin$ = "y" OR kakunin$ = "Y" THEN GOTO mondai.tukuri:
58:     IF kakunin$ = "n" OR kakunin$ = "N" THEN
59:         PRINT
60:         PRINT "やめちゃうの？　残念だなぁ　またやりましょうね"
61:         PRINT
62:         PRINT "バイバイ　・・・・"
63:     ELSE
64:         GOTO mouitido:
65:     END IF
66:
67: END
```

■プログラムの使い方

プログラムを実行すると、パソコンが問題をつくって、ゲームが開始されます。「さあ、いくつでしょうか」との問いに、キーボードから数字キーを押して答えます。答えとパソコンの考えている問題の数字をくらべて、大きいか小さいかがディスプレイに表示されます。この情報をもとにユーザーは次の回答を考えます。これを繰り返して、パソコンの考えている数字を当てるゲームです。

■プログラムの要点

17行　RANDOMIZEステートメントにTIMER関数を組み合わせて、時間によって発生する乱数系列を変えることにより、実行のたびに違った問題が出るようにしています。

18行　RND関数を使って0～1までの乱数を発生させて、INT関数で1～100の数に調整しています。

33～41行　変数mondaiとkaitouを比べて、回答が問題より大きいか小さいかを判断して、知らせています。

43行　変数mondaiとkaitouが等しければ正解ですので、ラベル名seikai:へ実行を移し、等しくなければkurikaesi:に実行を戻します。

46行　正解したときの案内をCOLORステートメントを使って水色で表示するように指定します。

集計計算プログラム　FOR～NEXTを使って

FOR～NEXTステートメントは、同じ作業を指定した回数繰り返す命令です。表集計では、FOR～NEXTステートメントを2つ組み合わせて、縦の行と横の桁の指定を次つぎと変化させて計算しています。

FOR～NEXT（フォー～ネクスト）ステートメント

書式　FOR　カウンタ変数＝初期値　TO　終了値　STEP　増分
　　　　実行ステートメント
　　NEXT
機能　一連の命令を指定された回数だけ繰り返して実行する

FOR～NEXTは、カウンタ変数を初期値から、終了値まで増分ずつ変化させて、FOR節とNEXT節の間の実行ステートメントをループ状に繰り返し実行する命令です。次のような順序で実行されます。

1．カウンタ変数はループカウンタ（繰り返す回数を計算するための変数）として使われ、最初に初期値に設定されます。

2．FORとNEXTに囲まれた範囲の実行ステートメントが実行されます。

3．カウンタ変数を増分だけ増減します。

4．カウンタ変数と終了値とを比較して、ループカウンタの値が終了値に達していなければ2．に戻り、同じ処理が繰り返されます。

5．カウンタ変数の値が終了値に達していれば、繰り返しを終えて、NEXTの次のステートメントに実行が移されます。

```
 1: '*****************************************************
 2: '                         集計計算
 3: '*****************************************************
 4: '
 5: 'shukei.bas                 ※左側の番号と：（コロン）
 6: '                             は入力しないでください
 7:     keta.jougen = 6
 8:     gyou.jougen = 20
 9:
10: hajime:
11:     CLS
12:     PRINT "集計計算をします （6桁・20行までです）"
13:     PRINT "何桁・何行 ありますか"
14:
15:     INPUT "横は 何桁"; keta
16:     INPUT "縦は 何行"; gyou
17:     IF keta.jougen < keta OR gyou.jougen < gyou THEN
18:         PRINT "6桁・20行までです もう一度どうぞ"
19:         PRINT "なにかキーを押してください"
20:         kari$ = INPUT$(1)
21:         GOTO hajime:
22:     END IF
23:
24:     DIM dat(keta, gyou), tate(keta), yoko(gyou)
25:
26: nyuryoku:
27:     PRINT "データを一行づつ入力してください"
28:     FOR i = 1 TO gyou
29:         PRINT i; "行"
30:         FOR j = 1 TO keta
31:             PRINT "    "; j; "桁";
32:             INPUT dat(j, i)
33:         NEXT
34:     NEXT
35:
36: keisan:
37:     '横計の計算
38:     FOR i = 1 TO gyou
39:         yoko(i) = 0
```

■プログラムの使い方

プログラムを実行すると、すぐに集計する表の桁数と、行数をたずねられます。最大6桁、20行までですので、キーボードから入力します。

大きすぎる値を入力すると、再度入力を求めてきますが、マイナスの値や文字などを入力しても、受け付けてしまうので、誤入力のないように注意してください。

次に、計算するデータを1行目から順に入力していきます。

すべてのデータを入力すると、計算を実行して、結果をディスプレイに表示して終了します。

```
40:         FOR j = 1 TO keta
41:             yoko(i) = yoko(i) + dat(j, i)
42:         NEXT
43:     NEXT
44:
45:     '縦計の計算
46:     FOR j = 1 TO keta
47:         tate(j) = 0
48:         FOR i = 1 TO gyou
49:             tate(j) = tate(j) + dat(j, i)
50:         NEXT
51:     NEXT
52:
53:     '総計の計算
54:     soukei = 0
55:     FOR j = 1 TO keta
56:         soukei = soukei + tate(j)
57:     NEXT
58:
59: hyouji:
60:     PRINT "計算しました     結果を表示します"
61:     '横の桁表示
62:     PRINT "     ";
63:     IF 10 < gyou THEN PRINT " ";
64:     FOR j = 1 TO keta
65:         PRINT USING "  #####桁"; j;
66:     NEXT
67:     PRINT "      合計"
68:
69:     '縦の行・入力データ・横計の表示
70:     FOR i = 1 TO gyou
71:         PRINT USING "##行  "; i;
72:         FOR j = 1 TO keta
73:             PRINT USING "######## "; dat(j, i);
74:         NEXT
75:         PRINT USING "######## "; yoko(i)
76:     NEXT
77:
78:     '縦計の表示
79:     PRINT
80:     PRINT "合計  ";
81:     FOR j = 1 TO keta
82:         PRINT USING "######## "; tate(j);
83:     NEXT
84:
85:     '総計の表示
86:     PRINT USING "######## "; soukei
87:
88: END
```

■プログラムの要点

7～8行　表の桁と、行の上限を設定しています。この数値を大きくすれば、大きな集計表の計算を行うことができます。

24行　データを格納する配列変数を宣言しています。dat( )は計算する個このデータを、tate( )は各桁の縦計を、yoko( )は各行の横計を格納する配列変数です。

26～34行　計算するデータを行ごとに入力します。FOR～NEXTステートメントの2重ループを使っています。

28～34行　外側のループで、カウンタ変数にiを使って行を数えています。

30～33行　内側のループで、カウンタ変数にjを使って桁を数えています。

36～57行　2重ループを使って、横計、縦計を計算し、総計の計算をしています。

59～88行　計算結果の表示を行います。

汎用メニュールーチン DO～LOOPを使って

実用プログラムの場合は、選択できるしごとの一覧を表示して、そのなかから目的の作業を指定できるようにすると簡単です。メニュー選択の際に、きちんと指定のキーが押されるまで、選択の操作を求めるというような、条件が整うまで同じことを繰り返すのにDO～LOOPステートメントを利用します。

DO～LOOP（ドゥ～ループ）ステートメント

書式
① DO　WHILEまたはUNTIL　条件式
　　実行ステートメント
　LOOP
② DO
　　実行ステートメント
　LOOP　WHILEまたはUNTIL　条件式

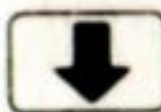
機能
①は先頭判断型、②は末尾判断型で、WHILEは条件が成立している間繰り返し、UNTILは条件が成立するまで、実行ステートメントを繰り返し実行する

DO～LOOPステートメントは、DOとLOOPの間の実行ステートメントを繰り返す命令です。繰り返しをつづけるか、中断するかの判断をループの先頭でするのが先頭判断型、ループの最後で行うのが末尾判断型です。条件判断に、条件が成立している間繰り返しを行うWHILE型の判断と、条件が成立していない間（つまり条件が成立するまで）繰り返すUNTIL型の判断があります。

```
1: '**************************************************
2: '                 汎用メニュールーチン
3: '**************************************************
```

※左側の番号と：（コロン）は入力しないでください

```
4:
5: 'menu.bas
6:
7: junbi:
8:
9:     DEFINT A-Z
10:
11:     kosu = 10
12:     Y = 1: Y = Y - 1
13:     x = 1
14:
15:     DIM menu$(kosu)
16:     FOR i = 1 TO kosu
17:         READ menu$(i)
18:     NEXT
19:
20:     ON KEY(11) GOSUB ue:
21:     ON KEY(14) GOSUB sita:
22:
23: hajime:
```

```
24:
25:     CLS
26:     FOR i = 1 TO kosu
27:         COLOR 7
28:         bango = i
29:         hyoujiketa = i + Y
30:         GOSUB menuhyouji:
31:     NEXT
32:
33:     KEY(11) ON
34:     KEY(14) ON
35:
36:     bango = 1
37:     DO
38:         COLOR 15
39:         hyoujiketa = bango + Y
40:         GOSUB menuhyouji:
41:
42:         DO
43:             ky$ = INKEY$
44:         LOOP WHILE ky$ = ""
45:
46:     LOOP WHILE ky$ <> CHR$(13)
47:
48:     KEY(11) OFF
49:     KEY(14) OFF
50:     COLOR 7
51:
52:     GOSUB sagyou:
53:
54: END
55:
56:
57: menuhyouji:
58:     LOCATE hyoujiketa, x
59:     PRINT USING "##"; bango;
60:     PRINT " : "; menu$(bango)
61:
62: RETURN
63:
64:
65: ue:
66:     COLOR 7
67:     hyoujiketa = bango + Y
68:     GOSUB menuhyouji:
69:
70:     bango = bango - 1
71:     IF bango < 1 THEN bango = kosu
72:     COLOR 15
73:     hyoujiketa = bango + Y
74:     GOSUB menuhyouji:
75:
76: RETURN
77:
78:
79: sita:
80:     COLOR 7
81:     hyoujiketa = bango + Y
82:     GOSUB menuhyouji:
83:
84:     bango = bango + 1
85:     IF kosu < bango THEN bango = 1
86:     COLOR 15
87:     hyoujiketa = bango + Y
88:     GOSUB menuhyouji:
89:
90: RETURN
91:
92:
93: sagyou:
94:     CLS
95:     PRINT "「"; bango; " 番のメニューが選ばれました」"
96:
97: RETURN
98:
99:
100: 'data
101:     DATA 1番のメニュー
102:     DATA 2番のメニュー
103:     DATA 3番のメニュー
104:     DATA 4番のメニュー
105:     DATA 5番のメニュー
106:     DATA 6番のメニュー
107:     DATA 7番のメニュー
108:     DATA 8番のメニュー
109:     DATA 9番のメニュー
110:     DATA 10番のメニュー
```

プログラムの使い方

プログラムをスタートすると、ディスプレイの左上に1〜10までのメニュー

が表示されます。メニュー選択は、カーソルキーの⬇⬆を使って、反転表示されている注目メニューを移動します。選択するメニューを反転表示にしたら、⏎キーを押します。メニューが選択されたことを示す「○番のメニューが選択されました」との案内表示が出て、終了します。

■プログラムの要点

9行　変数をすべて整数に宣言しています。

11行　変数kosuは、メニューの個数。この数字で選択枝のメニューを増減できます。

12〜13行　メニューの表示位置を変数yとxに設定します。

15〜18行　メニューに必要な数の配列を宣言して、配列変数menu$( )に読み込んでいます。

20〜21行　カーソルキーが押されたときのイベントトラッピング。

25〜31行　指定した位置にメニューを表示します。

33〜34行　イベントトラッピングを開始します。

37〜46行　このプログラムの本体の部分で、末尾判断型のDO〜LOOPステートメントを使っています。

48〜50行　ループから脱出すると、イベントトラッピングを解除して、色の指定も白色に戻しておきます。

52行　選択された作業を行うサブルーチン呼び出し。

54行　プログラムの終点を示します。これよりあとのプログラムは、メイン部から呼び出されるサブルーチンです。

100〜110行　メニューの内容を指定する文字列をデータとしてひとまとめにしておきます。ここに、本来のメニュー項目を指定します。

Yes／Noルーチン　SUBを使って

ユーザーからなにかの入力を求める場合、YまたはNでイエス、ノーを答えるものと、なにかのキーが押されるまで待つ単純なキー入力待ち、数字キーで選ぶ場合の3つの方法を取り上げたものです。

前の汎用メニュールーチンでは、いままでのBASICと同じようにサブルーチンを使いましたが、このプログラムではSUBプロシージャを使っています。こちらのほうがよりQuick BASICの機能を生かしたプログラムといえるでしょう。

SUB（サブ）ステートメント

書式 SUB プロシージャ名 パラメータ，パラメータ，………
機能 SUB プロシージャの定義を行う

SUB ステートメントは、メインプログラムから呼び出されるサブプログラムを定義するステートメントです。Quick BASIC のエディタ環境で SUB ステートメントを使うと、メインプログラムとは別のプログラムとして、サブプログラムが作成されます。

サブプログラムは、SUB から END SUB までが、指定したプロシージャ名で独立して管理されます。プロシージャ内では、メインプログラムと変数が独立して管理されていますので、同じ変数名を使ってもお互いに格納しているデータが変化してしまったりするといった影響は受けません。

変数が独立して管理されていても、パラメータを使って変数の内容を引数として、相互に引き渡すことができます。

```
1: DECLARE SUB yesno.sub (kari$)
2: DECLARE SUB mati.sub ()
3: DECLARE SUB sentaku.sub (kari$)
4: '*****************************************************
5: '              Ｙｅｓ／Ｎｏルーチン
6: '*****************************************************
7: '
8: 'yesno.bas
9: '
10:
11:     COMMON SHARED midori
12:     COMMON SHARED kiiro
13:     COMMON SHARED siro
14:
15:     midori = 2
16:     kiiro = 6
17:     siro = 7
18:
19: hajime:
20:     CLS
21:     COLOR kiiro
22:     PRINT "キーボードからの入力で処理を分けます"
23:     PRINT "さて、次の３つのうちどのサブプログラムを試しますか？"
24:     PRINT
25:     PRINT "Ｙｅｓ／Ｎｏルーチン－－－－１"
26:     PRINT "キー入力待ちルーチン－－－－２"
27:     PRINT "数字選択ルーチン－－－－－－３"
28:     PRINT
29:     INPUT "               何番にしますか"; kari$
```

※左側の番号と：（コロン）は入力しないでください

```
30:     kari = VAL(kari$)
31:     COLOR siro
32:
33:     SELECT CASE kari
34:         CASE 1
35:             yesno.sub kaisub$
36:         CASE 2
37:             mati.sub
38:         CASE 3
39:             CALL sentaku.sub(kaisub$)
40:         CASE ELSE
41:             PRINT "1から3までの数字で入力してください"
42:     END SELECT
43:
44:     COLOR kiiro
45:     PRINT
46:     PRINT "'main'プログラムに戻りました"
47:     COLOR siro
48:     PRINT
49:     PRINT "プログラムを終了します"
50:
51: END
52:
53: SUB mati.sub
54:     COLOR midori
55:     PRINT
56:     PRINT "'mati.sub'サブプログラムに来ました"
57:     COLOR siro
58:
59:     PRINT
60:     PRINT "なにかキーを押してください"
61:     kari$ = ""
62:     DO WHILE kari$ = ""
63:         kari$ = INKEY$
64:     LOOP
65:
66:     COLOR midori
67:     PRINT
68:     PRINT "キーが押されました"
69:     COLOR siro
70: END SUB
71:
72: SUB sentaku.sub (kari$)
73:     COLOR midori
74:     PRINT
75:     PRINT "'sentaku.sub'サブプログラムに来ました"
76:     COLOR siro
77:
78:     PRINT
79:     PRINT "0 から 9 の数字で選んでください"
80:     kari$ = ""
81:     DO WHILE kari$ < "0" OR "9" < kari$
82:         kari$ = INPUT$(1)
```

■プログラムの使い方

実行すると、1～3のキーで答えるようにメニューが表示されます。

1・・・YまたはNでイエス、ノーを答える。

2・・・キーが押されるまで待っている、単純なキー入力待ち。

3・・・数字キーを使って0～9を選ぶ。

3つの方法から実験してみたいルーチンを数字の1～3のキーを押して選択します。

1．Yes／No選択は、キーのYまたはNを押します。それ以外のキーを押しても受け付けてくれません。

2．キー入力待ちは、なにかのキーを押すまではなにもせずに、キーが押されるのを待ちます。

3．数字選択は、キーボードの0～9の数字キーが押されると、押された数字を表示しますが、他のキーは受け

```
 83:     LOOP
 84:
 85:     COLOR midori
 86:     PRINT
 87:     PRINT kari$; "が選ばれました"
 88:     COLOR siro
 89:
 90:     kaisub$ = kari$
 91: END SUB
 92:
 93: SUB yesno.sub (kari$)
 94:     COLOR midori
 95:     PRINT
 96:     PRINT "'yesno.sub'サブプログラムに来ました"
 97:     COLOR siro
 98:
 99:     PRINT
100:     PRINT "よろしいですか?  (はい='Y'es / いいえ='N'o ) "
101:     kari$ = ""
102:     DO WHILE kari$ = ""
103:         kari$ = INPUT$(1)
104:         IF kari$ = "y" OR kari$ = "Y" THEN
105:             kaisub$ = "y"
106:         ELSEIF kari$ = "n" OR kari$ = "N" THEN
107:             kaisub$ = "n"
108:         ELSE
109:             kari$ = ""
110:         END IF
111:     LOOP
112:
113:     COLOR midori
114:     PRINT
115:     PRINT kari$; "が選ばれました"
116:     COLOR siro
117: END SUB
```

付けてくれません。

1は「はい」「いいえ」の回答を求める場合に、2はメッセージなどを表示して読むまでの間の時間待ちに、3はメニューなどでの数字を使った選択に利用するためのルーチンです。

■プログラムの要点

このプログラムは、1～51行のメインプログラムと、53～70行のmati. sub、72～91行のsenta-ku. sub、93～117行のyesno. subの3つのサブプログラムからできています。

3つのサブプログラムは、それぞれキー入力待ち、数字選択、Yes／No選択のしごとをする独立したプロシージャで、プログラムの先頭で宣言されています。

1～3行　SUBプログラムの宣言をしています。

11～13行　COMMON SHAREDステートメントを使って、変数の共用を宣言しています。

22～28行　3つのサブプログラムのうちから試したいものを選択するようにメッセージを表示しています。

29～30行　変数kari$に選択した番号を入力して、VAL関数によって文字列として入力された数字を数値に変えています。

33～42行　選ばれた番号で、それぞれSUBプログラムを呼び出しています。

40～41行　1～3以外の数字や文字列が入力された場合に、案内をしています。必ず条件外の措置をCASE ELSE節に指定しておきます。

58～69行　キー入力待ちのプロシージャです。

61～64行　変数kari$を空にして、DO～LOOPステートメントを使ったループに入っています。INKEY$関数を使って、キーボードからの入力を受け付けます。

72～91行　数字選択のプロシージャです。

80～83行　このプロシージャの中心部分です。INPUT$(1)関数で、キーボードからの1文字入力を受け付けます。

93～117行　Yes／No選択のプロシージャです。

101～111行　このプロシージャの中心部分です。INPUT$(1)で1文字が入力された結果、Yまたはyの場合、Nまたはnの場合に限って受け付けています。

時間計算プログラム　FUNCTIONを使って

なにかの経過時間、つまり初めの時間から、終わりの時間まで何時間何分何秒だったか、すぐに計算するプログラムです。

FUNCTION（ファンクション）ステートメント

書式　FUNCTION　プロシージャ名　（パラメータ，パラメータ………）
機能　FUNCTION　プロシージャの定義を行う

FUNCTIONは、メインプログラムから呼び出されるサブプログラムを定義します。Quick BASICのエディタ環境でFUNCTIONを使うと、メインプログラムとは別のプログラムとして、サブプログラムが作成されます。

サブプログラムは、FUNCTIONからEND FUNCTIONまでが、指定したプロシージャ名で管理され、ひとつのプロシージャとして独立します。

プロシージャ内では、メインプログラムと変数が独立して管理されているので、同じ変数名を使ってもお互いに格納しているデータが変化してしまうという影響は受けません。

FUNCTIONプロシージャは、SUBプロシージャと違って、ステートメントのように呼び出すのではなく、計算や評価を行う式のなかで関数と同様に呼び出して使用します。

※左側の番号と：（コロン）
は入力しないでください

```
 1: DECLARE FUNCTION jikan$ (kari)
 2: DECLARE FUNCTION byou (kari$)
 3: '***************************************************
 4: '                  時間計算プログラム
 5: '***************************************************
 6: '
 7: 'jikan.bas
 8: '
 9: hajime:
10:     CLS
11:     PRINT "初めから終わりまでの経過時間を計算します（１日以内です）"
12:     PRINT "時間は「00:00:00～23:59:59」の２４時間制で入力してください"
13:     PRINT
14:     PRINT "初めの時間を  「時時：分分：秒秒」でどうぞ"
15:     INPUT kaisi$
16:     PRINT "終わりの時間を「時時：分分：秒秒」でどうぞ"; ""
17:     INPUT shuryou$
18:
19:     IF shuryou$ < kaisi$ THEN
20:         PRINT "初めの時間は終わりの時間より前でなければなりません"
21:         INPUT "リターンキーを押してください", dummy$
22:         GOTO hajime:
23:     END IF
24:
25:     keika = byou(shuryou$) - byou(kaisi$)
26:
27:     PRINT jikan$(keika); "経過"
28:
29: END
30:
31:
32:
33: FUNCTION byou (kari$)
34:
35:     kji = VAL(LEFT$(kari$, 2)) * 3600
36:     khun = VAL(MID$(kari$, 4, 2)) * 60
37:     kbyou = VAL(RIGHT$(kari$, 2))
38:
39:     byou = kji + khun + kbyou
40:
41: END FUNCTION
42:
43: FUNCTION jikan$ (kari)
44:     kji = kari ¥ 3600
45:     khun = (kari - kji * 3600) ¥ 60
46:     kbyou = (kari - kji * 3600) - khun * 60
47:
48:     kji$ = STR$(kji) + "時間"
49:     khun$ = STR$(khun) + "分"
50:     kbyou$ = STR$(kbyou) + "秒"
51:
52:     jikan$ = kji$ + khun$ + kbyou$
53:
54: END FUNCTION
```

プログラムの使い方

プログラムをスタートすると、画面に注意事項を表示します。時間を計算するために、初めの時間を入力しますが、時間は「00：00：00～23：59：59」の24時間制で、時分秒の間の区切りとして「：」(コロン)を使用してキーボードから入力します。次に、同じように終わりの時間を入力しますが、範囲は24時間内です。入力が終わると、計算結果の経過時間を表示して終了します。

計算できる範囲を1日以上にしたり、データを一括して入力するなどして、自分自身のタイムマネジメントに使えるプログラムに発展させてみましょう。

プログラムの要点

9～29行　メインプログラムです。多くが入力のためのチェック部分です。

10～17行　画面をクリア。注意事項を表示して、データの入力を求めます。

19行　開始時間と終了時間を文字列のまま、アスキーコードで比較しています。

21行　INPUT で変数 dummy$に入力を求めていますが、必要なデータがあるわけではなく、メッセージを読むまでの時間かせぎをしています。

22行　条件が成立すると、プログラムを単純化するために直接9行の hajime：へ処理を移します。

25行　FUNCTION プロシージャ byou を呼び出して、経過時間を計算します。

27行　FUNCTION プロシージャ jikan$を呼び出して、経過時間を「時時：分分：秒秒」の文字列に直します。

33～41行　byou プロシージャです。パラメータに kari$を使って、メインプログラムと引数をやり取りしています。引数の内容は時間を示す文字列です。

35～37行　kari$に受け取った時間の文字列データを数値データに変えてから、秒に換算しています。

39行　FUNCTION プロシージャの名まえ byou に、換算の結果を計算して代入しています。プロシージャ名に結果を代入して、引数をメインプログラムに引き渡します。

43～54行　FUNCTION プロシージャ jikan$です。

44～46行　秒で表わされた時間を時分秒の単位に換算しています。

48～50行　換算した時間を文字列に変えて、時間、分、秒の単位を示す文字を付加しています。

52行　プロシージャ名 jikan$に時間分秒のデータを代入して、メインにデータを引き渡します。

パソコンで表現しよう

ニュートンのりんご　LOCATEを使って

簡単なステートメントを使ったアニメーションです。テレビアニメのようにはいきませんが、自分でつくったプログラムが動くなんて、とっても楽しいものです。

「ニュートンのりんご」と、ちょっと気取った題名をつけてみました。題名は「パリの枯葉」でもよいのですが、簡単な計算とステートメントを組み合わせて、小さなメルヘンを表現しました。

LOCATE（ロケート）ステートメント

書式　LOCATE　行，列，カーソル，上端，下端

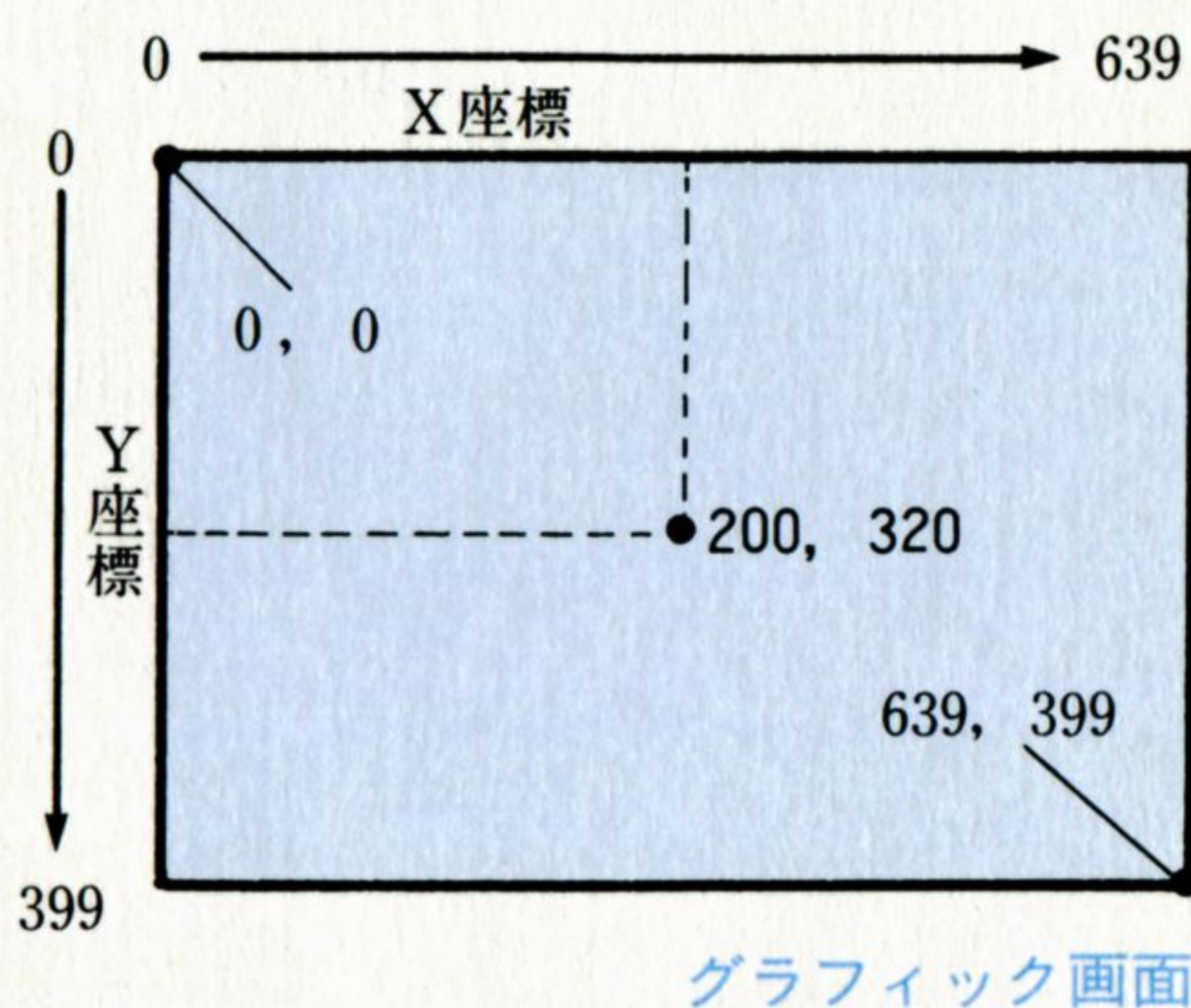
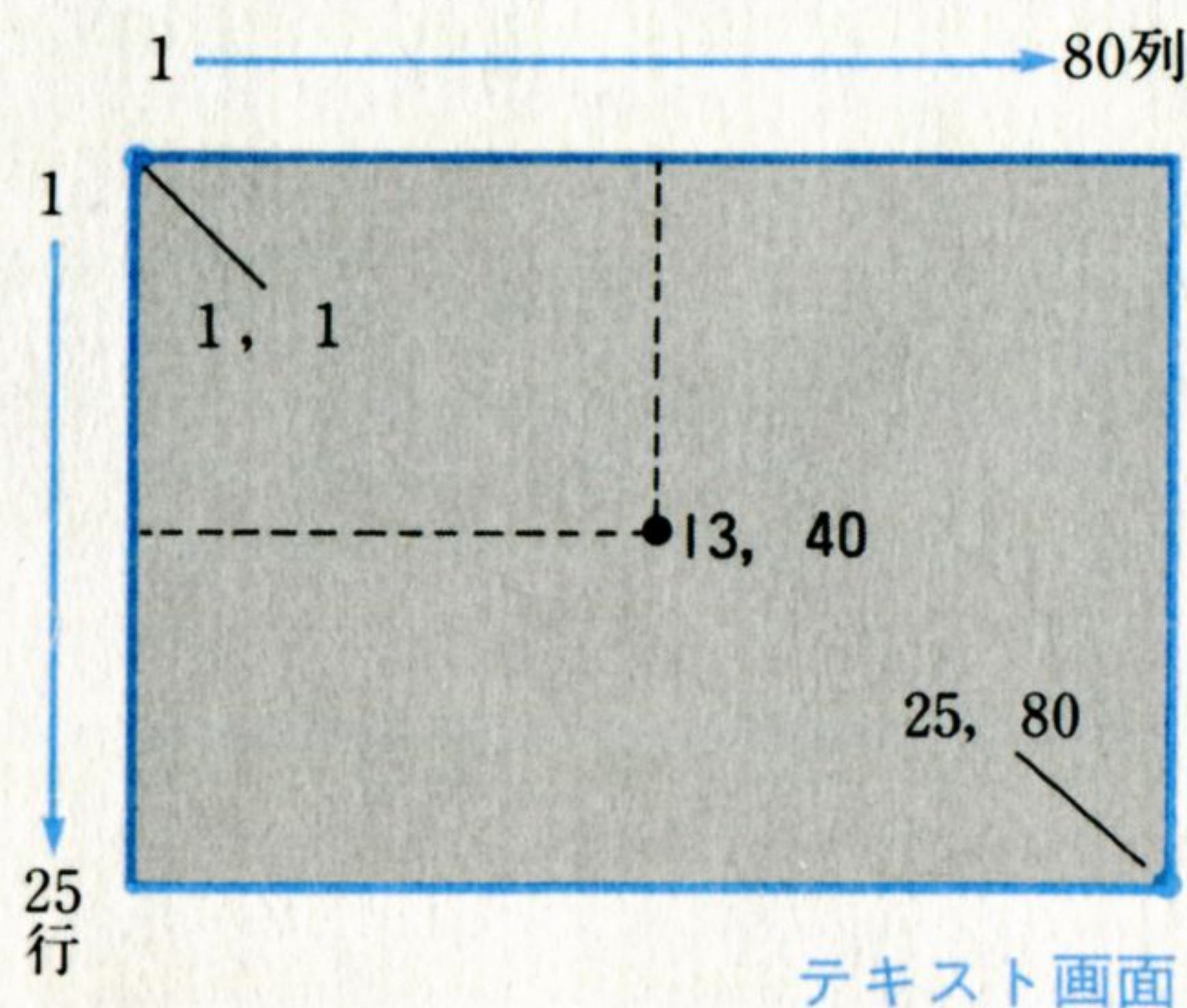
機能　カーソルを指定した位置に移動する

Quick BASICでのディスプレイ画面は、文字を表示する80列×25行のテキスト画面と、絵を表示する640×400ピクセル、または640×200ピクセルのグラフィック画面に分かれています。テキスト画面は左上の端を「1，1」（1行，1列）、右下を「25，80」とする座標で、画面上の位置を指定することができます。グラフィック画面は水平にX座標、垂直にY座標をとります。

1　80列
1　25行
1，1
13，40
25，80

テキスト画面

0　639
X座標
0　Y座標　399
0，0
200，320
639，399

グラフィック画面

LOCATEは、テキスト画面上のどの位置に文字を表示するかを指定します。カーソルは、画面上の四角く点滅している点で、文字はカーソルの位置から表示されます。

このカーソルをLOCATEステートメントで移動して、PRINTステートメントなどによる文字の表示位置を指定します。LOCATEとPRINTは、表示位置を指定して文字を表示するというように組み合わせて使用します。

```
 1: '*************************************************
 2: '              ニュートンのりんごプログラム
 3: '*************************************************
 4: '                                 ※左側の番号と：（コロン）
 5: 'ringo.bas                          は入力しないでください
 6: '
 7: hajime:
 8:     CLS
 9:     COLOR 2
1Ø:     LOCATE 1, 1Ø
11:
12:     PRINT "      ¦ /-------- /----/-----/                 "
13:     PRINT "      /  ¦ -----/                              "
14:     PRINT "¦    /   /   /                                 "
15:     PRINT "¦    ¦  /----                                  "
16:     PRINT " ¦ / /                                         "
17:     PRINT " ¦ ¦/                                          "
18:     PRINT "  ¦ /                                          "
19:     PRINT "  ¦¦                                           "
2Ø:     PRINT "  ¦¦                                           "
21:     PRINT "  ¦¦                                           "
22:     PRINT "  ¦¦                                           "
23:     PRINT "  ¦¦                                           "
24:     PRINT "  ¦¦                                           "
25:     PRINT "  ¦¦                                           "
26:     PRINT "  ¦¦                                           "
27:     PRINT "  ¦¦                                           "
28:     COLOR 6
29:     PRINT "-/ ¦---------------------------------------"
3Ø:
31:     FOR x = 1Ø TO 4Ø STEP 4
32:         COLOR 2
33:         LOCATE 2, x
34:         PRINT "o";
35:     NEXT
36:
37:
38:     FOR x = 1Ø TO 4Ø STEP 4
39:         LOCATE 2, x
4Ø:         PRINT " ";
41:
42:         FOR y = 2 TO 16
43:
44:             ookisa = y - 1
45:             LOCATE y, x, 1, Ø, ookisa
46:             FOR j = 1 TO 5ØØ: NEXT
47:
```

■**プログラムの使い方**

プログラムをスタートすれば、あとは見ているだけです。りんごの木から、実が落ちてるように見えれば、あなたの想像力はすばらしいものです。枯葉がひらひらと落ちるように改造してみるのも楽しいでしょう。

■**プログラムの要点**

10行　カーソルを1行、10列に移動します。

12〜27行　りんごの木のイメージをPRINTで表示します。

28〜29行　色を黄色に変えて、りんごの木の根と地面を表現します。

31〜35行　FOR〜NEXTループを使ってりんごを表示します。

33〜34行　LOCATEの第2オプションをループカウンタを使って移動しては、小文字のo（オー）をりんごの実に見立てます。

```
48:         NEXT
49:
5Ø:         COLOR 4
51:         LOCATE 17, x
52:         PRINT "Ø";
53:
54:     NEXT
55:
56:     LOCATE 23, 1, 1, Ø, 15
57:     COLOR 7
58:
59: END
```

38～54行　2つのループを入れ子にして、りんごの実が次つぎと落ちていくようすを表現します。外側のループは、左から順番に落ちるりんごの実を指定しています。

39～40行　指定されたりんごの実を画面から消しています。

42～48行　内側のループ。りんごが落ちていく様子です。

44～45行　変数 ookisa を使って、地面に近づいたりんごの実が大きくなっていくように見せます。

46行　FOR～NEXT ステートメントを使った時間稼ぎのためのループです。こうしないと、人間には目にも止まらないスピードで、表示されてしまいます。

50～52行　地上に落ちたりんごの実を数字の0で表現しています。

56～57行　プログラムを終了する前に、LOCATE と COLOR の指定を標準に戻します。

星空プログラム　PSETを使って

グラフィック画面は、テキスト画面とは独立して点や線、円や箱型（ボックス）などを描けます。高解像度ディスプレイなら、1画面に水平640×垂直400で25万6千、標準でも640×200で12万8千もの点を描くことができます。

この点がピクセルですが、画面のランダムな位置のピクセルをランダムな色で、ひとつずつ描きこんでいくと、きれいな星空ができあがってきます。

PSET（ポイントセット）ステートメント

> **書式**　PSET　（X座標，Y座標），カラー番号
> **機能**　グラフィック画面に点を表示する

グラフィック画面の自由な位置に絵を表示させる最も基本的なステートメントがPSETです。座標は基本的にディスプレイの左上の端が（0，0）、右下の端が（639，399）になります。水平に右にいくほど増えていくのがX座標で、垂直に下にいくほど増えるのがY座標です。

```
 1: '****************************************
 2: '                        星空プログラム
 3: '****************************************
 4: '
 5: 'hosizora.bas
 6: '
 7: junbi:
 8:     SCREEN 87
 9:     WINDOW SCREEN (0, 0)-(639, 399)
10:     VIEW SCREEN (0, 0)-(639, 399)
11:     CLS 0
12:
13: hajime:
14:     RANDOMIZE TIMER
15:
16:     DO
17:         x = INT(RND(1) * 639) + 1
18:         y = INT(RND(1) * 399) + 1
19:         iro = INT(RND(1) * 7) + 1
20:
21:         PSET (x, y), iro
22:
23:     LOOP WHILE INKEY$ = ""
24:
25: END
```

※左側の番号と：（コロン）は入力しないでください

プログラムの使い方

プログラムをスタートすると、画面に夜空の星のように色とりどりの点が表示されます。なにかのキーを押すとプログラムを終了します。

プログラムの要点

8行　グラフィックモードを640×400のスーパーインポーズモードで使う指定をします。

9行　グラフィック画面の論理座標を実際にディスプレイに表示される座標の物理座標と同じ座標系に合わせています。

10行　ビューポートをディスプレイ上の物理座標に合わせています。

14行　乱数の発生系列を初期化。

17行　RND関数とINT関数を利用して、1～639までの整数値を発生して、X座標を指定するために変数xに代入しています。

18行　同様に1～399までの整数値を発生して、変数yに代入します。

19行　色指定のために使うカラー番号を同じように1～7までのランダムな整数を発生して、変数iroに代入しています。

21行　ランダムにつくったx，yの点を座標指定に、iroをカラー番号指定に利用して、星に見立てた点を表示しています。

棒グラフ　LINEを使って

棒グラフは、たくさんあるグラフのなかでも基本中の基本で、量を比較したり、推移を見るのに適しています。棒グラフをつくるには、グラフィックで線を引くLINEを利用します。

LINE（ライン）ステートメント

書式　LINE　（始点X，始点Y）－（終点X，終点Y），色，箱

機能　グラフィック画面に線や箱型を表示する

グラフィック画面上の２点間を結ぶ直線を引くステートメントです。始点と終点はそれぞれX座標、Y座標で指定します。

```
 1: '****************************************************
 2: '                    棒グラフプログラム
 3: '****************************************************
 4: '                            ※左側の番号と：（コロン）
 5: 'bograph.bas                   は入力しないでください
 6: '
 7: junbi:
 8:     SCREEN 87
 9:     WINDOW SCREEN (0, 0)-(639, 399)
10:     VIEW SCREEN (0, 0)-(639, 399)
11:     CLS 0
12:
13:     datasuu = 10
14:     DIM bodata(datasuu)
15:     FOR i = 1 TO datasuu
16:         READ bodata(i)
17:     NEXT
18:
19:     DATA 83,72,43,52,92,78,30,71,55,89
20:
21: hajime:
22:     LOCATE 1, 30
23:     PRINT "棒   グ   ラ   フ"
24:     LINE (80, 16)-(584, 224), 2, B
25:     FOR i = 24 TO 204 STEP 20
26:         LINE (78, i)-(82, i), 2
27:     NEXT
28:     LOCATE 2, 6: PRINT USING "####"; 100
29:     LOCATE 8, 6: PRINT USING "####"; 50
30:     LOCATE 14, 6: PRINT USING "####"; 0
31:
32:     FOR i = 1 TO datasuu
33:         iro = INT(i MOD 7) + 1
34:         LINE ((i * 48) + 57, 224 - (bodata(i) * 2))-((i * 48) + 80, 224),iro, BF
35:         LOCATE 15, i * 6 + 9
36:         PRINT CHR$(i + &H40);
37:     NEXT
38:
39:     FOR i = 1 TO datasuu / 2
40:         LOCATE i + 17, 10
41:         PRINT CHR$(i + &H40); "  "; bodata(i)
42:
43:         LOCATE i + 17, 40
44:         PRINT CHR$(i + &H40 + 5); "  "; bodata(i + 5)
45:     NEXT
46:
47: END
```

棒グラフプログラムは10個までの与えられたデータを棒グラフにして表示します。データを棒グラフにするこつは、データの数値をいかにグラフィック画面の座標に換算するかです。

基準の線を定めて、データの値が大きいほどX座標の値が減っていくようにします。座標が決定したあとは、LINEの箱型オプションを使って、基準線とデータの値に従った座標の点を結ぶ棒を表示します。

■プログラムの使い方

プログラムをスタートすると、19行で与えられたデータを棒グラフにして表示します。

データを変えたり、データの個数を変えて試してみてください。

■**プログラムの要点**

8～11行　グラフィック画面の設定と、画面のクリア。

13行　グラフ化するデータの数を変数datasuuに格納します。

14行　データ数分の配列を用意するための配列変数の宣言。

15～17行　配列にデータを読み込みます。

19行　グラフ化するデータをDATAステートメントを使って与えます。

32～37行　データ数分の棒グラフを表示します。

34行　LINEステートメントに色と箱（BF）を指定して棒を表示します。

円グラフ　CIRCLEを使って

円グラフは全体の合計を100として、個別の数値がそれに占める割合を百分率で計算して表現するグラフです。

百分率が出れば、円の360°に換算して、簡単に円グラフにすることができます。

CIRCLE（サークル）ステートメント

書式	CIRCLE　（中心X，中心Y），半径，色，開始角，終了角
機能	グラフィック画面に円、円弧、扇型を表示する

中心と半径を定めて、グラフィック画面に円、円弧、扇型を描くステートメントです。中心の位置は、中心Xと中心Yを座標で指定し、半径はピクセル数で指定します。

※左側の番号と：（コロン）は入力しないでください

```
 1: '**************************************************
 2: '                 円グラフプログラム
 3: '**************************************************
 4: '
 5: 'engraph.bas
 6: '
 7: junbi:
 8:     SCREEN 87
 9:     WINDOW SCREEN (0, 0)-(639, 399)
10:     VIEW SCREEN (0, 0)-(639, 399)
11:     CLS 0
12:
13:     CONST Pai = 3.141592
14:
15:     DataSuu = 6
16:     DIM Kaisha$(DataSuu)
17:     DIM EnData(DataSuu)
```

■**プログラムの使い方**

プログラムをスタートすると、27～28行に与えられたデータを円グラフにします。データを変えたり、データの個数を変えれば、いろいろな円グラフができます。

■**プログラムの要点**

8～11行　グラフィッ

パソコンで表現しよう

```
18:      DIM Iro(DataSuu)
19:      DIM Wariai(DataSuu)
20:
21:      FOR i = 1 TO DataSuu
22:          READ Kaisha$(i)
23:          READ EnData(i)
24:          READ Iro(i)
25:      NEXT
26:
27:      DATA 日本商事,939,4,東京物産,421,6,大阪商店,256,3
28:      DATA 名古屋KK,208,1,札幌産業,168,2,そ の 他,411,5
29:
30:
31: hajime:
32:      LOCATE 2, 11
33:      PRINT "円   グ   ラ   フ"
34:
35:      FOR i = 1 TO DataSuu
36:          Goukei = Goukei + EnData(i)
37:      NEXT
38:
39:      FOR i = 1 TO DataSuu
40:          Wariai(i) = EnData(i) / Goukei
41:      NEXT
42:
43:      KaisiKaku = Pai / 2
44:
```

ク画面の設定です。

13行　CONSTを使って、変数Paiに定数として3.141592を代入しています。以後は、Paiを使って角度の計算ができます。

16〜19行　データ数分の配列を用意するために、配列変数の宣言を行います。

21〜25行　それぞれの配列にデータを読み込みます。

27〜28行　グラフ化するデータを与えます。

31〜62行　このプログラムの本体部分です。

```
45:      FOR i = DataSuu TO 1 STEP -1
46:          ShuuryouKaku = KaisiKaku + Wariai(i) * Pai * 2
47:          IF Pai * 2 < ShuuryouKaku THEN ShuuryouKaku = ShuuryouKaku - Pai * 2
48:          CIRCLE (160, 160), 112, Iro(i), -KaisiKaku, -ShuuryouKaku
49:          KaisiKaku = ShuuryouKaku
50:      NEXT
```

```
51:
52:      FOR i = 1 TO DataSuu
53:          LOCATE i * 2 + 4, 40
54:          COLOR Iro(i)
55:          PRINT Kaisha$(i);
56:          COLOR 7
57:          PRINT "   "; EnData(i);
58:          PRINT USING "  (###.##"; Wariai(i) * 100;
59:          PRINT "%)"
60:      NEXT
61:
62: END
```

45〜50行　円グラフを表示しています。FOR〜NEXTでデータを小さいものから順に表示しています。

52〜60行　各社の売り上げデータと、シェアを%で表示します。

ディスクを活用しよう

電話帳プログラム　OPEN／INPUT　#／WRITE　#を使って

シーケンシャルファイルを使った電話帳プログラムです。さて、あなたの電話帳にはいったい何人の電話番号が載っていますか？　このプログラムを応用していくと、きっとすべての人の電話番号をディスクに記録して、便利に利用できるようになるでしょう。

OPEN（オープン）ステートメント

書式　OPEN　ファイル名　FOR　モード　AS　#ファイル番号
機能　ファイルを使えるように設定する

OPENは、フロッピーディスク上のファイルを使えるようにするため、指定したファイルの読み出し、書き込みのモードを設定します。

INPUT　#（インプット　ナンバー）ステートメント

書式　INPUT　#ファイル番号，変数
機能　シーケンシャルファイルからデータを読み出し、変数に入れる

INPUT　#は、OPENステートメントでオープンされたフロッピーディスクに記録されているファイル番号のシーケンシャルファイルからデータを読み出します。読み込まれたデータは、指定された変数に格納されます。

WRITE　#（ライト　ナンバー）ステートメント

書式　WRITE　#ファイル番号，データ
機能　シーケンシャルファイルにデータを書き込む

WRITE　#は、OPENステートメントでフロッピーディスク上にオープンされたファイル番号のシーケンシャルファイルにデータを書き込みます。書き込まれたデータとデータの間は、「，」（カンマ）で自動的に区切られます。

同じような働きをするステートメントにPRINT　#（プリント　ナンバー）がありますが、これは区切り記号を付加しませんので、プログラムのなかで区切りを指定しなければなりません。

プログラムの使い方

このプログラムは、電話帳をパソコンに管理させます。スタートすると、メインメニューが表示されます。ひとつのしごとを終わったり、異常な入力が行われて、処理を続けられなくなると、このメインメニューに戻ります。

メニューは、キーボードの1～7の数字キーを押して選択します。

1　追加登録　データを追加します。
2　修　　正　登録されているデータを修正します。
3　削　　除　必要のなくなったデータを消します。
4　検　　索　該当するデータを探し出します。
5　一覧表示　登録されているデータを順番に表示します。
6　一覧印刷　範囲を指定してデータの一覧を印刷します。
7　終　　了　このプログラムを終了します。

1　追加登録

シメイ　氏名の読みをカタカナで入力。名まえからの検索に利用する。⏎キーだけ押すとメインメニューに戻る。
氏　名　漢字の名まえを入力する。
摘　要　家族や勤務先、趣味などなんでも記録できる。
TEL　電話番号を入力。つづいてFAX番号なども記録できる。
・3つの項目を順番に入力。
・カーソルキーで、他の項目へは移れない。
・データのない項目は、⏎キーだけを押す。
・最後の確認で訂正のある場合は「Y」キーを押す。「N」を押すと、データの登録を終了してメインメニューに戻る。

2　修　　正

修正するデータの番号を入力すると、現在登録されている内容が画面に表示されるので、順番にデータを修正する。
・カーソルキーで、他の項目の入力には移れない
・データの変更のない項目は、先頭で⏎キーだけを押す。
・一部を変更した場合は、登録するデータの最後の文字の右側まで、カーソル移動してから⏎キーを押す。
・最後の確認で訂正のある場合は「Y」キーを押して、訂正する。「N」を押すと、訂正されたデータを登録してメインメニューに戻る。

3　削　　除

必要のなくなったデータは、データの番号を指定して削除する。何番のデータを削除するか、その番号を入力する。
・削除してよければ「Y」を、まちがって選択して削除しないなら「N」を押す。
・削除が終わるとメインメニューに戻る。

4　検　　索

名まえや読み、摘要に記録されている事項をもとに該当するデータを探し出す。検索する名まえや摘要に記録されている家族や勤務先などのキーデータを入力する。
・該当するデータが、見つかった場合は、データの全項目を表示する。
・「続けてデータを検索します」と表示されたら、「Y」を押すと、同じキーデータで検索を続行する。
・これ以上検索をつづける必要のない場合は、「N」キーを押して中断する。
・検索を終わり「すべてのデータを検索し終わりました」と表示されたら、なにかキーを押すとメインメニューに戻る。

5　一覧表示

データを1番から順番に表示する。
・「E」キーを押すと一覧表示を中止して、メインメニューに戻る。

・「E」以外のキーを押すと、表示が一時停止し、データの項目を読むことができる。
・再度スペースバーを押すと表示を続行。

6 一覧印刷

何番のデータから、何番のデータまでを印刷するか範囲を指定する。初めの番号が小さく、終わりが大きくなければならない。
・番号の指定が終わり「プリンタの準備ができたらなにかキーを押してください」と案内が出たら、なにかキーを押す。
・印刷を途中で中止したい場合は「E」キーを押す。
・印刷し終わるか、Eキーが押された場合は、メインメニューに戻る。

7 終 了

登録されているデータをすべてフロッピーディスクに記録して、プログラムを終了する。

※左側の番号と：（コロン）は入力しないでください

```
 1: '*****************************************************
 2: '                    電話帳プログラム
 3: '*****************************************************
 4: '
 5: 'denwa.bas
 6: '
 7: junbi:
 8:     CONST Iie = 0
 9:     CONST Hai = NOT Iie
10:     CONST Saidai = 1000
11:     CONST MenuKosuu = 7
12:     CONST KoumokuSuu = 4
13:     SaidaiBangou = 0
14:     Bangou = 0
15:
16:     DIM TelData$(KoumokuSuu, Saidai)
17:     DIM Menu$(MenuKosuu)
18:     DIM Koumoku$(KoumokuSuu)
19:
20:     GOSUB DataSettei:
21:     OPEN "DENWA.DAT" FOR APPEND AS #1
22:     CLOSE
23:     GOSUB DataYomikomi:
24:
25: Hajime:
26:     CLS
27:     DO
28:         GOSUB Sentaku:
29:         SELECT CASE MenuBangou
30:             CASE 1
31:                 GOSUB Touroku:
32:             CASE 2
33:                 GOSUB Shuusei:
34:             CASE 3
35:                 GOSUB Sakujo:
36:             CASE 4
37:                 GOSUB Kensaku:
38:             CASE 5
39:                 GOSUB Itiran:
40:             CASE 6
41:                 GOSUB Insatu:
42:             CASE 7
```

■プログラムの要点

（メインルーチン）

7～23行　プログラムの本体に入る前の準備。

8～9行　プログラムのなかで使う条件判断「はい」「いいえ」を1と0の定数として設定。

10～12行　データの最大個数を1000、メニューの個数を7、データの項目数を4に設定。

13～14行　データの番号を管理するための変数をゼロクリアします。

16～18行　データ、メニュー、データの項目を格納する配列を宣言。

20行　データの設定は、サブルーチンとして独立させて呼び出します。

21～22行　ダミーのOPENステートメントです。ファイル名DENWA.DATをオ

```
43:                 GOSUB Shuuryou:
44:             CASE ELSE
45:                 COLOR 6
46:                 LOCATE MenuKosuu * 2 + 4, 21
47:                 PRINT "1～7の番号で選択してください"
48:                 COLOR 7
49:                 Kari$ = INPUT$(1)
50:         END SELECT
51:     LOOP WHILE Shuuryou = Iie
52:     CLS
53: END
54:
55: '-------------------- サブルーチン ------------------
56:
57: Sentaku:
58:     CLS
59:     LOCATE 1, 28
60:     PRINT "電　話　帳"
61:     FOR i = 1 TO MenuKosuu
62:         LOCATE i * 2 + 2, 25
63:         PRINT Menu$(i); " : "; i
64:     NEXT
65:     LOCATE MenuKosuu * 2 + 4, 21
66:     PRINT "番号で選択してください : ";
67:     Kaitou$ = INPUT$(1)
68:     MenuBangou = VAL(Kaitou$)
69: RETURN
70:
71: Touroku:
72:     SaidaiBangou = SaidaiBangou + 1
73:     Bangou = SaidaiBangou
74:     Teisei$ = "y"
75:         CLS
76:         LOCATE 1, 1
77:         PRINT Menu$(MenuBangou); " No."; Bangou
78:         GOSUB DataHyouji:
79:         FOR i = 1 TO KoumokuSuu
80:             LOCATE i * 2 + 1, 8
81:             INPUT "", Kotae$
82:             TelData$(i, Bangou) = Kotae$
83:             IF TelData$(1, Bangou) = "" THEN
84:                 SaidaiBangou = SaidaiBangou - 1
85:                 RETURN
86:             END IF
87:         NEXT
88:         LOCATE MenuKosuu * 2 + 4, 1
89:         PRINT "訂正がありますか ( 'Y'es / 'N'o )"
90:         Teisei$ = LCASE$(INPUT$(1))
```

ープンして、データファイルがフロッピーディスクになかった場合は、新たにファイルを作成します。

23行　ファイルからデータを読み出すサブルーチンを呼び出します。

25～53行　プログラムの本体。メニューで選択したしごとによってSELECT～CASEステートメントを使って処理を分岐させ、終了が選択されるまでループして繰り返します。

（サブルーチン）

55～226行　プロシージャを使わずに、普通のサブルーチンとして、メインプログラムだけからなるプログラムにしています。

（選択サブルーチン）

57～69行　しごとのメニューを表示して、キーボードから数字キーで選択します。

（208ページにつづく）

```
91:     IF Teisei$ = "y" OR Teisei$ = "ｙ" THEN GOSUB DataTeisei:
92: RETURN
```

```
 93:
 94: Shuusei:
 95:     CLS
 96:     LOCATE 1, 1
 97:     PRINT Menu$(MenuBangou); "  No.1 ～"; "  No."; SaidaiBangou
 98:     PRINT
 99:     INPUT "何番のデータを修正しますか "; Kari$
1ØØ:     Bangou = VAL(Kari$)
1Ø1:     IF Bangou < 1 OR SaidaiBangou < Bangou THEN RETURN
1Ø2:     GOSUB DataTeisei:
1Ø3: RETURN
1Ø4:
1Ø5: Sakujo:
1Ø6:     CLS
1Ø7:     LOCATE 1, 1
1Ø8:     PRINT Menu$(MenuBangou); "  No.1 ～"; "  No."; SaidaiBangou
1Ø9:     PRINT
11Ø:     INPUT "何番のデータを削除しますか "; Kari$
111:     Bangou = VAL(Kari$)
112:     IF Bangou < 1 OR SaidaiBangou < Bangou THEN RETURN
113:     Teisei$ = "y"
114:     CLS
115:     LOCATE 1, 1
116:     PRINT Menu$(MenuBangou); "  No."; Bangou
117:     GOSUB DataHyouji:
118:     LOCATE MenuKosuu * 2 + 4, 1
119:     COLOR 4
12Ø:     PRINT "このデータを削除します"
121:     COLOR 7
122:     PRINT "良いですね ( 'Y'es / 'N'o )"
123:     Kakunin$ = LCASE$(INPUT$(1))
124:     IF Kakunin$ = "y" OR Kakunin$ = "ン" THEN
125:         FOR i = 1 TO KoumokuSuu
126:             TelData$(i, Bangou) = ""
127:         NEXT
128:     END IF
129: RETURN
13Ø:
131: Kensaku:
132:     CLS
133:     LOCATE 1, 1
134:     PRINT Menu$(MenuBangou); "  No.1 ～"; "  No."; SaidaiBangou
135:     PRINT
136:     INPUT "検索する文字列は "; KensakuMoji$
137:     FOR i = 1 TO SaidaiBangou
138:         FOR j = 1 TO KoumokuSuu
139:             IF INSTR(TelData$(j, i), KensakuMoji$) <> Ø THEN
14Ø:                 Bangou = i
141:                 CLS
142:                 LOCATE 1, 1
143:                 PRINT Menu$(MenuBangou); "  No."; Bangou
144:                 GOSUB DataHyouji:
145:                 LOCATE MenuKosuu * 2 + 4, 1
```

```
146:                PRINT "続けてデータを検索します"
147:                PRINT "良いですね ( 'Y'es / 'N'o )"
148:                Kakunin$ = LCASE$(INPUT$(1))
149:                IF Kakunin$ = "n" OR Kakunin$ = "ミ" THEN RETURN
150:            END IF
151:        NEXT
152:    NEXT
153:    LOCATE MenuKosuu * 2 + 4, 1
154:    PRINT "すべてのデータを検索し終わりました"
155:    PRINT "メニューに戻ります何かキーを押してください"
156:    Kakunin$ = INPUT$(1)
157: RETURN
158:
159: Itiran:
160:    VIEW PRINT
161:    CLS
162:    LOCATE 1, 1
163:    PRINT Menu$(MenuBangou); "  No.1 ～"; "  No."; SaidaiBangou
164:    LOCATE 24, 1
165:    PRINT "'スペース'で続行    'E'で中止"
166:    VIEW PRINT 3 TO 23
167:    DO
168:        Chuusi = Iie
169:        i = 0
170:        DO
171:            i = i + 1
172:            PRINT "------"; i; "------"
173:            FOR j = 1 TO KoumokuSuu
174:                PRINT Koumoku$(j); " : ";
175:                PRINT TelData$(j, i)
176:            NEXT
177:            PRINT
178:            Kari$ = INPUT$(1)
179:            IF LCASE$(Kari$) = "e" OR Kari$ = "イ" THEN Chuusi = Hai
180:            IF SaidaiBangou <= i THEN Chuusi = Hai
181:        LOOP UNTIL Chuusi = Hai
182:    LOOP UNTIL LCASE$(Kari$) = "e" OR Kari$ = "イ"
183:    VIEW PRINT
184: RETURN
185:
186: Insatu:
187:    CLS
188:    LOCATE 1, 1
189:    PRINT Menu$(MenuBangou); "  No.1 ～"; "  No."; SaidaiBangou
190:    PRINT
191:    PRINT "何番から何番を印刷しますか"
192:    INPUT "印刷開始"; InsatuKaisi
193:    IF InsatuKaisi < 1 OR SaidaiBangou < InsatuKaisi THEN RETURN
194:    INPUT "    終了"; InsatuShuuryou
195:    IF InsatuShuuryou < 1 OR SaidaiBangou < InsatuShuuryou THEN RETURN
196:    PRINT
197:    PRINT "プリンタの準備ができたら何かキーを押してください。"
198:    Kari$ = INPUT$(1)
```

```
199:     LOCATE 24, 1
200:     PRINT "'E'で印刷中止"
201:     VIEW PRINT 3 TO 23
202:     Chuusi = Iie
203:     i = InsatuKaisi
204:     DO
205:         PRINT "------"; i; "------"
206:         LPRINT "------"; i; "------"
207:         FOR j = 1 TO KoumokuSuu
208:             PRINT Koumoku$(j); " : ";
209:             PRINT TelData$(j, i)
210:             LPRINT Koumoku$(j); " : ";
211:             LPRINT TelData$(j, i)
212:         NEXT
213:         PRINT
214:         LPRINT
215:         Kari$ = INKEY$
216:         i = i + 1
217:         IF LCASE$(Kari$) = "e" OR Kari$ = "イ" THEN Chuusi = Hai
218:         IF InsatuShuuryou <= i THEN Chuusi = Hai
219:     LOOP UNTIL Chuusi = Hai
220:     VIEW PRINT
221: RETURN
222:
223: Shuuryou:
224:     GOSUB DataKakikomi:
225:     Shuuryou = Hai
226: RETURN
227:
228: '------------------ 作業用サブルーチン ------------------
229:
```

```
230: DataSettei:
231:     FOR si = 1 TO MenuKosuu
232:         READ Menu$(si)
233:     NEXT
234:     DATA データの 追加登録
235:     DATA データの 修   正
236:     DATA データの 削   除
237:     DATA データの 検   索
238:     DATA データの 一覧表示
239:     DATA データの 一覧印刷
240:     DATA -------- 終   了
241:
242:     FOR si = 1 TO KoumokuSuu
243:         READ Koumoku$(si)
244:     NEXT
245:     DATA "ｼﾒｲ "
246:     DATA 氏名
247:     DATA 摘要
248:     DATA "TEL "
249: RETURN
250:
```

67〜68行　キーボードからの入力を文字列として受け、VAL関数を使って数値にして、変数MenuBangouに選択された番号を格納します。

(登録サブルーチン)

71〜92行　データを入力するためのサブルーチン。

72〜73行　これから入力するデータの番号を最大にしています。

74〜91行　データ入力のための一組のブロック。訂正がなければ、このブロックを抜けます。

79〜86行　データを入力。81行のINPUT

```
251: DataHyouji:
252:     LOCATE 3, 1
253:     FOR si = 1 TO KoumokuSuu
254:         PRINT Koumoku$(si); " : ";
255:         COLOR 3
256:         PRINT TelData$(si, Bangou)
257:         COLOR 7
258:         PRINT
259:     NEXT
260: RETURN
261:
262: DataTeisei:
263:     Teisei$ = "y"
264:     DO
265:         CLS
266:         LOCATE 1, 1
267:         PRINT Menu$(MenuBangou); "  No."; Bangou
268:         GOSUB DataHyouji:
269:         FOR si = 1 TO KoumokuSuu
270:             LOCATE si * 2 + 1, 8
271:             INPUT "", Kotae$
272:             IF Kotae$ <> "" THEN
273:                 TelData$(si, Bangou) = Kotae$
274:             END IF
275:         NEXT
276:         LOCATE MenuKosuu * 2 + 4, 1
277:         PRINT "訂正がありますか ( 'Y'es / 'N'o )"
278:         Teisei$ = LCASE$(INPUT$(1))
279:     LOOP WHILE Teisei$ = "y" OR Teisei$ = "ｙ"
280: RETURN
281:
282: DataYomikomi:
283:     OPEN "DENWA.DAT" FOR INPUT AS #1
284:     i = 0
285:     DO UNTIL EOF(1)
286:         i = i + 1
287:         FOR j = 1 TO KoumokuSuu
288:             INPUT #1, TelData$(j, i)
289:         NEXT
290:     LOOP
291:     CLOSE
292:     SaidaiBangou = i
293: RETURN
294:
295: DataKakikomi:
296:     OPEN "DENWA.DAT" FOR OUTPUT AS #1
297:     FOR i = 1 TO SaidaiBangou
298:         FOR j = 1 TO KoumokuSuu
299:             WRITE #1, TelData$(j, i)
300:         NEXT
301:     NEXT
302:     CLOSE
303: RETURN
```

で入力を求め、82行でデータを配列に格納しています。

83〜86行　カナ氏名のデータになにも入力されなかった場合は、メインルーチンに戻ります。

88〜91行　データに訂正がないかどうかの確認。訂正があれば訂正のサブルーチンを呼び出します。

90行　LCASE$関数で、1文字入力された文字を小文字に変換して91行での条件判断を簡単にしています。

（修正サブルーチン）

94〜103行　修正するデータの番号を入力して、データ修正サブルーチンを呼び出すサブルーチンです。

101行　番号の入力まちがいを調べて、まちがえていたときは、メインルーチンに戻します。

（削除サブルーチン）

105〜129行　不要になったデータを削除します。

106〜112行　削除するデータの番号の入力を求めます。

113〜123行　データの削除について確認を求めます。

124〜128行　Yキーが押されるとデータを削除します。

（検索サブルーチン）

131～157行　入力された検索文字列がデータのなかにないかを検索するサブルーチンです。

136行　検索したい文字列の入力を求めます。

137～152行　二重のFOR～NEXTループ。外側のループで1～最大番号までの全データを順番に探します。内側のループで配列の各要素のデータと検索文字列を比較しています。

139行　INSTR関数で検索文字列がデータに含まれていないかを調べています。

140～149行　検索文字列が見つかった場合の確認の処理です。

144行　見つかったデータの番号を140行で設定して、データ表示サブルーチンを呼び出して、内容を表示します。

146～149行　検索をつづける確認。Nが選ばれると、メインルーチンに戻ります。

153～156行　検索を終わると、メインルーチンに戻ります。

（一覧サブルーチン）

159～184行　登録されているデータを順番に表示するサブルーチン。

160行　テキスト画面のすべてを表示範囲とする画面の初期化です。

167～182行　二重のDO～LOOP。外側のループは表示の中止、続行の判断、内側のループは表示データの指定と各項目データの表示を行います。

171行　表示データを指定する番号を、カウントアップしています。

172～177行　データを表示しています。

178～180行　Eが入力されたときとループカウンタiが最大番号を越えたときに、181行でのループ脱出の準備のために、変数ChuusiをHaiにしています。このように、ある状態になったことをプログラムの他の部分に知らせることを、「フラグを立てる」といいます。

181行　変数ChuusiがHaiでフラグが立っていれば、内側のループを抜けます。

182行　E以外のキーが押された場合は、ループを終了せず、Eが押されるとループを抜けます。

（印刷サブルーチン）

186～226行　データの一覧印刷をするサブルーチンです。

191〜195行　印刷を開始するデータの番号と、終了するデータの番号を入力します。それぞれ1より小さくないか、開始の番号が最大の番号を越えていないかを確かめて、異常な値を入力された場合、メインルーチンに戻って、メニュー選択からやり直します。

203〜219行　一覧サブルーチンをもとに、エディタ機能を利用して、印刷のための行を加えるなどの修正を行い、効率的にプログラムします。

（終了サブルーチン）

223〜226行　終了のサブルーチンです。

224行　プログラムの終了の前に、登録されているデータを書き込みます。

225行　変数 Shuuryou に Hai を代入して、プログラムを終了するようにフラグを立ててメインルーチンへ知らせます。

（作業用サブルーチン）

228〜303行　メインルーチンから各しごとをする専門のサブルーチンを呼び出し、またそのサブルーチンから呼び出されて細かい作業を行う「サブサブルーチン」ともいえるサブルーチンの集まりです。

（データ設定サブルーチン）

231〜249行　READ〜DATA を使って、配列にメニューの作業項目名やデータの項目名を設定します。

（データ表示サブルーチン）

252〜260行　データの項目と内容を表示するサブルーチンです。

253行　呼び出したルーチンと変数名がかち合わないように、FOR〜NEXT のループカウンタ変数に si を使っています。サブルーチン用のカウンタ変数という意味から、Sがつけ加えられています。

（データ訂正サブルーチン）

263〜280行　データの内容を表示して、まちがいのある項目のデータを入力して訂正するサブルーチンです。

272〜274行　なんの入力もなく、⏎キーが押されると、変数 Kotae$には空の文字が格納されます。空文字でなければ、訂正のデータが入力されたことになるので、配列 TelData$( )に入力されたデータを代入します。

（データ読み込みサブルーチン）

283〜293行　フロッピーディスクに記録されたシーケンシャルデータファイル DENWA.DAT から、配列にデータを読み出します。

283行　ファイルをデータ読み出しのためにオープンします。

285～290行　DO～LOOPの判断にEOF関数を使って、ファイルの最後になったらループを抜けて、データの読み出しを終わるようにしています。

288行　ファイルからデータを読み出して、配列TelData$( )に格納します。

291行　使用を終わったらファイルはすぐにクローズしておきます。

(データ書き込みサブルーチン)

296～303行　入力されたデータをフロッピーディスクに記録します。

296行　ファイルを書き込みのためにオープンします。

297～301行　FOR～NEXTのループカウンタの終了値をデータの最大番号にして、1番から最大番号までのデータすべてを記録します。

298～300行　内側のループは、データの項目を配列の要素ごとに記録します。

299行　配列TelData$( )に格納されているデータを書き込みます。

302行　使用を終わったファイルはすぐにクローズします。

名刺ホルダープログラム

TYPE～END TYPE／GET #／PUT #を使って

名刺ホルダープログラムは、住所録や人物データベースにも応用できるプログラムです。このプログラムでは、氏名だけではなく、会社名や所属、役職などのすべての項目と、摘要に記録された自由なデータで検索ができます。

TYPE～END TYPE（タイプ～エンド タイプ）ステートメント

書式　TYPE ユーザーデータ型名
　　　　要素名　AS　型名
　　　　　　：
　　　END TYPE

機能　ユーザー定義データ型を定義する

GET #　（ゲット　ナンバー）ステートメント

書式　GET　#ファイル番号，レコード番号，変数

機能　フロッピーディスク上のランダムファイルから指定されたレコード番号のデータを変数に読み出す

PUT # (プット ナンバー) ステートメント

書式 PUT #ファイル番号, レコード番号, 変数
機能 フロッピーディスク上のランダムファイルの指定されたレコード番号に、変数にセットされているデータを書き込む

■プログラムの使い方

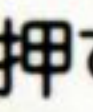
このプログラムは、名刺をパソコンに管理させるプログラムですが、住所録や人物データベースとしても利用することができます。

プログラムをスタートすると、メインメニューが表示されます。ひとつのしごとを終わったり、異常な入力があって、処理をつづけられなくなると、このメニューに戻ります。

メニューは、キーボードの数字キーを押して選択します。

1 追加登録 データを追加します。
2 修　　正 登録されているデータを修正します。
3 削　　除 必要のなくなったデータを消します。
4 検　　索 該当するデータを探し出します。
5 一覧表示 登録されているデータを順番に表示します。
6 一覧印刷 範囲を指定してデータの一覧を印刷します。
7 終　　了 このプログラムを終了します。

1 追加登録

シメイ 氏名の読みをカタカナで入力する。名まえからの検索に利用する。
氏　名 漢字の名まえを入力する。
会　社 勤務先の名称を入力する。
所　属 勤務先の所属を入力する。
役　職 勤務先での役職を入力する。
TEL 電話番号を入力する。会社や自宅の電話番号を自由に記録できる。
FAX ＦＡＸ番号を入力する。
摘　要 名刺を交換した時間や場所だけでなく、この人の印象や趣味などなんでも記録できる。

・各項目を順番に入力する。
・シメイの入力で、⏎キーだけを押すと、メインメニューに戻ることができる。
・カーソルキーで、他の項目の入力に移ることはできない。
・データのない項目は、⏎キーだけを押す。
・データ入力が終わり、訂正があるかどうか聞かれたら、訂正のある場合は「Y」キーを押す。「N」を押すと、データの登録を終了してメインメニューに戻る。

2 修　正

修正するデータの番号を入力すると、現在登録されている内容が画面に表示されるので、項目ごとに順番にデータを修正する。

・カーソルキーで、他の項目の入力に移ることはできない。
・データに変更のない項目は、データの先頭で⏎キーだけを押す。
・データの一部を変更した場合は、登録するデータの最後の文字の右側まで、カーソル移動してから⏎キーを押して登録する。
・データ入力が終わり、訂正があるかどう

かたずねられたら、訂正のある場合は「Y」キーを押す。「N」を押すと、修正されたデータを登録してメインメニューに戻る。

3 削　除

必要のなくなったデータは、データの番号を指定して削除する。何番のデータを削除するか聞かれたら、番号を入力する。

・削除してよければ「Y」を、削除したくない場合は「N」を押す。
・削除が終わると、メインメニューに戻る。

4 検　索

氏名や会社名、摘要に記録されている事項をもとに該当するデータを探し出す。検索する文字列をたずねられたら、探したい名まえや摘要に記録されている家族や勤務先などのキーデータを入力する。

該当するデータを探して、見つかったら、データの全項目を表示する。

・「続けてデータを検索します」と表示され、「Y」を押すと検索を続行する。
・目的のデータが見つかったら「N」キーを押して検索を中断する。
・検索終了の表示があったとき、なにかキーを押すとメインメニューに戻る。

5 一覧表示

登録されているデータを順に表示する。

・「E」を押すと、メインメニューに戻る。
・ E 以外のキーを押すと、表示を一時停止するので、データの項目を読むことができる。
・再度スペースバーを押すと表示を続行。

6 一覧印刷

まず何番のデータから、何番のデータを印刷するか範囲を指定する。番号は初めが小さく、終わりが大きくなければならない。

・番号の指定が終わり、プリンタの準備ができたら、なにかキーを押す。
・印刷を途中で中止したい場合は、「E」キーを押す。
・印刷し終わるか、Eキーが押された場合は、メインメニューに戻る。

7 終　了

・7を選択すると、プログラムを終了する。

```
 1: DECLARE FUNCTION SENTAKU! ()
 2: DECLARE SUB TOUROKU ()
 3: DECLARE SUB SHUUSEI ()
 4: DECLARE SUB SAKUJO ()
 5: DECLARE SUB KENSAKU ()
 6: DECLARE SUB ITIRAN ()
 7: DECLARE SUB INSATU ()
 8: DECLARE SUB SHUURYOU ()
 9: DECLARE SUB DATA.SETTEI (MK, KS)
10: DECLARE SUB DATA.HYOUJI (D$(), KS)
11: DECLARE SUB DATA.TEISEI (D$(), B, KS)
12: DECLARE SUB DATA.YOMIKOMI (D$(), B)
13: DECLARE SUB DATA.KAKIKOMI (D$(), B)
14: '*****************************************************
15: '                     名刺ホルダープログラム
16: '*****************************************************
17: '
18: 'meisistr.bas
19: '
20: junbi:
21:     CONST Iie = 0
22:     CONST Hai = NOT Iie
23:     CONST Saidai = 1000
24:     CONST MenuKosuu = 7
25:     CONST KoumokuSuu = 8
26:
```

※左側の番号と：（コロン）は入力しないでください

■プログラムの要点

このプログラムは、メインプログラムと、メインプログラムから選択されたしごとに従って呼び出されるサブプログラム、および専門の作業を行うサブプログラムからなるプロシージャで構成されるシングルモジュールプログラムです。

（メインプログラム）

1～13行　プロシージャの宣言部。Quick BASICのエディタ環

```
27:     DIM SHARED SaidaiBangou
28:     DIM SHARED Bangou
29:     DIM SHARED DATA$(KoumokuSuu)
30:     DIM SHARED Menu$(MenuKosuu)
31:     DIM SHARED Koumoku$(KoumokuSuu)
32:     DIM SHARED Owari
33:
34:     SaidaiBangou = 0
35:     Bangou = 0
36:
37:     TYPE meisi
38:         Simei AS STRING * 20
39:         KanjiSimei AS STRING * 20
40:         Kaisha AS STRING * 40
41:         Shozoku AS STRING * 40
42:         Yakushoku AS STRING * 40
43:         Tel AS STRING * 15
44:         Fax AS STRING * 15
45:         Tekiyou AS STRING * 80
46:     END TYPE
47:     DIM SHARED MeisiData AS meisi
48:
```

境でプログラムした場合、自動的に付加されます。

21〜25行　定数を変数名とともに宣言します。

27〜32行　モジュール内で共通に使用する変数を共用変数として宣言します。

34〜35行　データの番号を管理する変数をゼロクリアしています。

37〜47行　ユーザー定義データ型を定義して

```
49:     OPEN "meisistr.dat" FOR RANDOM AS #1 LEN = LEN(MeisiData)
50:     SaidaiBangou = LOF(1) ¥ LEN(MeisiData)
51:     CLOSE
```

```
52:
53:     DATA.SETTEI MenuKosuu, KoumokuSuu
54:
55: Main:
56:     DO
57:     CLS
58:     MenuBangou = SENTAKU
59:         CLS
60:         SELECT CASE MenuBangou
61:             CASE 1
62:                 TOUROKU
63:             CASE 2
64:                 SHUUSEI
65:             CASE 3
66:                 SAKUJO
67:             CASE 4
68:                 KENSAKU
69:             CASE 5
70:                 ITIRAN
71:             CASE 6
72:                 INSATU
73:             CASE 7
74:                 SHUURYOU
```

います。定義された型で変数を宣言します。

49行　OPENを使って、ファイル名meisistr.datをランダムモードでオープン。

50行　LOF関数を使って求めたデータファイルの長さを変数の長さで割り、値をレコード数にします。

53行　メニューやデータの項目を設定するプロシージャを呼び出しています。

```
 75:                 CASE ELSE
 76:                     COLOR 6
 77:                     LOCATE MenuKosuu * 2 + 4, 21
 78:                     PRINT "1～7の番号で選択してください"
 79:                     COLOR 7
 80:                     Kari$ = INPUT$(1)
 81:             END SELECT
 82:     LOOP WHILE Owari = Iie
 83:     CLS
 84: END
 85:
 86: MenuData:
 87:     DATA データの　追加登録
 88:     DATA データの　修　　正
 89:     DATA データの　削　　除
 90:     DATA データの　検　　索
 91:     DATA データの　一覧表示
 92:     DATA データの　一覧印刷
 93:     DATA --------　終　　了
 94:
 95: KoumokuData:
 96:     DATA "ｼﾒｲ "
 97:     DATA 氏名
 98:     DATA 会社
 99:     DATA 所属
100:     DATA 役職
101:     DATA "TEL "
102:     DATA "Fax "
103:     DATA 摘要
104:
105: SUB DATA.HYOUJI (D$(), KS)
106:     LOCATE 3, 1
107:     FOR i = 1 TO KS
108:         PRINT Koumoku$(i); " : ";
109:         COLOR 3
110:         PRINT D$(i)
111:         COLOR 7
112:         PRINT
113:     NEXT
114: END SUB
115:
116: SUB DATA.KAKIKOMI (D$(), B)
117:     MeisiData.Simei = D$(1)
118:     MeisiData.KanjiSimei = D$(2)
119:     MeisiData.Kaisha = D$(3)
120:     MeisiData.Shozoku = D$(4)
121:     MeisiData.Yakushoku = D$(5)
122:     MeisiData.Tel = D$(6)
123:     MeisiData.Fax = D$(7)
124:     MeisiData.Tekiyou = D$(8)
125:     OPEN "meisistr.dat" FOR RANDOM AS #1 LEN = LEN(MeisiData)
```

55～84行　プログラムの本体です。選択のFUNCTIONプロシージャを用いて、しごとの番号を選びます。

SELECT～CASEを使って対応するプロシージャを呼び出し、終了になるまでループして繰り返します。

86～103行　DATAステートメントを使って、メニューとデータ項目の文字列データを定義します。

（プロシージャ）

Quick BASICでは、プロシージャの表示や印刷の順番の管理をアルファベット順にしています。

（データ表示SUBプロシージャ）

105～114行　データの項目と内容を表示するSUBプロシージャ。

107行　FOR～NEXTのループカウンタ変数にiを使っています。メイン部と同じ変数名

```
126:     RecordNumber = B
127:     PUT #1, RecordNumber, MeisiData
128:     CLOSE
129: END SUB
130:
131: SUB DATA.SETTEI (MK, KS)
132:     RESTORE MenuData:
133:     FOR i = 1 TO MK
134:         READ Menu$(i)
135:     NEXT
136:
137:     RESTORE KoumokuData:
138:     FOR i = 1 TO KS
139:         READ Koumoku$(i)
140:     NEXT
141: END SUB
142:
143: SUB DATA.TEISEI (D$(), B, KS)
144:     Teisei$ = "y"
145:     DO
146:         CLS
147:         LOCATE 1, 1
148:         PRINT Menu$(MenuBangou); "  No."; B
149:         RecordNumber = B
150:         DATA.YOMIKOMI D$(), B
151:         DATA.HYOUJI D$(), KS
152:         FOR i = 1 TO KS
153:             LOCATE i * 2 + 1, 8
154:             INPUT "", Kotae$
155:             IF Kotae$ <> "" THEN
156:                 D$(i) = Kotae$
157:             END IF
158:         NEXT
159:         DATA.KAKIKOMI D$(), B
160:         LOCATE KS * 2 + 3, 1
161:         PRINT "訂正がありますか ( 'Y'es / 'N'o )"
162:         Teisei$ = LCASE$(INPUT$(1))
163:     LOOP WHILE Teisei$ = "y" OR Teisei$ = "ソ"
164: END SUB
165:
166: SUB DATA.YOMIKOMI (D$(), B)
```

でも影響されません。

(データ書き込みSUBプロシージャ)

116～129行　変数に格納されているデータをランダムファイルに書き込みます。

117～124行　パラメータの配列を使って引き渡された引数のデータを、ランダムファイル用変数に格納します。

125～128行　ファイルをオープンし、ランダムファイルにデータを書き込んでいます。終了したらファイルをクローズします。

(データ設定SUBプロシージャ)

131～141行　READ～DATAで、配列にメニューの作業項目名やデータの項目名を設定します。

```
167:     OPEN "meisistr.dat" FOR RANDOM AS #1 LEN = LEN(MeisiData)
168:     RecordNumber = B
169:     GET #1, RecordNumber, MeisiData
170:     CLOSE
171:     D$(1) = MeisiData.Simei
172:     D$(2) = MeisiData.KanjiSimei
173:     D$(3) = MeisiData.Kaisha
174:     D$(4) = MeisiData.Shozoku
175:     D$(5) = MeisiData.Yakushoku
176:     D$(6) = MeisiData.Tel
```

```
177:     D$(7) = MeisiData.Fax
178:     D$(8) = MeisiData.Tekiyou
179: END SUB
180:
181: SUB INSATU
182:     CLS
183:     LOCATE 1, 1
184:     PRINT Menu$(MenuBangou); "  No.1 ～"; "  No."; SaidaiBangou
185:     PRINT
186:     PRINT "何番から何番を印刷しますか"
187:     INPUT "印刷開始"; InsatuKaisi
188:     IF InsatuKaisi < 1 OR SaidaiBangou < InsatuKaisi THEN EXIT SUB
189:     INPUT "    終了"; InsatuShuuryou
190:     IF InsatuShuuryou < 1 OR InsatuShuuryou < InsatuKaisi THEN EXIT SUB
191:     PRINT
192:     PRINT "プリンタの準備ができたら何かキーを押してください"
193:     Kari$ = INPUT$(1)
194:     LOCATE 24, 1
195:     PRINT "'E'で印刷中止"
196:     VIEW PRINT 3 TO 23
197:     Chuusi = Iie
198:     Bangou = InsatuKaisi
199:     DO
200:         RecordNumber = Bangou
201:         DATA.YOMIKOMI DATA$(), RecordNumber
202:         LPRINT "------"; Bangou; "------"
203:         FOR i = 1 TO KoumokuSuu
204:             LPRINT Koumoku$(i); " : ";
205:             LPRINT DATA$(i)
206:         NEXT
207:         LPRINT
208:         Kari$ = INKEY$
209:         IF LCASE$(Kari$) = "e" OR Kari$ = "ｲ" THEN Chuusi = Hai
210:         IF InsatuShuuryou <= Bangou THEN Chuusi = Hai
211:         Bangou = Bangou + 1
212:     LOOP UNTIL Chuusi = Hai
213:     VIEW PRINT
214: END SUB
215:
216: SUB ITIRAN
217:     VIEW PRINT
218:     CLS
219:     LOCATE 1, 1
220:     PRINT Menu$(MenuBangou); "  No.1 ～"; "  No."; SaidaiBangou
221:     LOCATE 24, 1
222:     PRINT "'スペース'で続行    'E'で中止"
223:     VIEW PRINT 3 TO 23
224:     DO
225:         Chuusi = Iie
226:         Bangou = 0
227:         DO
228:             Bangou = Bangou + 1
229:             RecordNumber = Bangou
```

```
230:            DATA.YOMIKOMI DATA$(), RecordNumber
231:            PRINT "------"; Bangou; "------"
232:            FOR i = 1 TO KoumokuSuu
233:                PRINT Koumoku$(i); " : ";
234:                PRINT DATA$(i)
235:            NEXT
236:            PRINT
237:            Kari$ = INPUT$(1)
238:            IF LCASE$(Kari$) = "e" OR Kari$ = "イ" THEN Chuusi = Hai
239:            IF SaidaiBangou <= Bangou THEN Chuusi = Hai
240:        LOOP UNTIL Chuusi = Hai
241:    LOOP UNTIL LCASE$(Kari$) = "e" OR Kari$ = "イ"
242:    VIEW PRINT
243: END SUB
244:
245: SUB KENSAKU
246:    CLS
247:    LOCATE 1, 1
248:    PRINT Menu$(MenuBangou); "  No.1 ～"; "  No."; SaidaiBangou
249:    PRINT
250:    INPUT "検索する文字列は "; KensakuMoji$
251:    FOR i = 1 TO SaidaiBangou
252:        Bangou = i
253:        RecordNumber = Bangou
254:        DATA.YOMIKOMI DATA$(), RecordNumber
255:        FOR j = 1 TO KoumokuSuu
256:            IF INSTR(DATA$(j), KensakuMoji$) <> 0 THEN
257:                CLS
258:                LOCATE 1, 1
259:                PRINT Menu$(MenuBangou); "  No."; Bangou
260:                DATA.HYOUJI DATA$(), KoumokuSuu
261:                LOCATE KoumokuSuu * 2 + 3, 1
262:                PRINT "続けてデータを検索します"
263:                PRINT "良いですね ( 'Y'es / 'N'o )"
264:                Kakunin$ = LCASE$(INPUT$(1))
265:                IF Kakunin$ = "n" OR Kakunin$ = "ミ" THEN EXIT SUB
266:            END IF
267:        NEXT
268:    NEXT
269:    LOCATE KoumokuSuu * 2 + 3, 1
270:    PRINT "すべてのデータを検索し終わりました"
271:    PRINT "メニューに戻ります何かキーを押してください"
272:    Kakunin$ = INPUT$(1)
273: END SUB
274:
275: SUB SAKUJO
276:    CLS
277:    LOCATE 1, 1
278:    PRINT Menu$(MenuBangou); "  No.1 ～"; "  No."; SaidaiBangou
279:    PRINT
280:    INPUT "何番のデータを削除しますか "; Kari$
281:    Bangou = VAL(Kari$)
282:    IF Bangou < 1 OR SaidaiBangou < Bangou THEN EXIT SUB
```

```
283:     Teisei$ = "y"
284:     CLS
285:     LOCATE 1, 1
286:     PRINT Menu$(MenuBangou); "  No."; Bangou
287:     RecordNumber = Bangou
288:     DATA.YOMIKOMI DATA$(), RecordNumber
289:     DATA.HYOUJI DATA$(), KoumokuSuu
290:     LOCATE KoumokuSuu * 2 + 3, 1
291:     COLOR 4
292:     PRINT "このデータを削除します"
293:     COLOR 7
294:     PRINT "良いですね ( 'Y'es / 'N'o )"
295:     Kakunin$ = LCASE$(INPUT$(1))
296:     IF Kakunin$ = "y" OR Kakunin$ = "ｙ" THEN
297:         FOR i = 1 TO KoumokuSuu
298:             DATA$(i) = ""
299:         NEXT
300:         RecordNumber = Bangou
301:         DATA.KAKIKOMI DATA$(), RecordNumber
302:     END IF
303: END SUB
304:
305: FUNCTION SENTAKU
306:     FOR i = 1 TO KoumokuSuu
307:         DATA$(i) = ""
308:     NEXT
309:     LOCATE 1, 28
310:     PRINT "名刺ホルダー"
311:     FOR i = 1 TO MenuKosuu
312:         LOCATE i * 2 + 2, 25
313:         PRINT Menu$(i); "  : "; i
314:     NEXT
315:     LOCATE MenuKosuu * 2 + 4, 21
316:     PRINT "番号で選択してください  :  ";
317:     Kaitou$ = INPUT$(1)
318:     SENTAKU = VAL(Kaitou$)
319: END FUNCTION
320:
321: SUB SHUURYOU
322:     Owari = Hai
323: END SUB
324:
325: SUB SHUUSEI
326:     CLS
327:     LOCATE 1, 1
```

（データ訂正SUBプロシージャ）

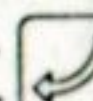
143〜164行　まちがいのある項目のデータを訂正するサブルーチンです。

152〜158行　なにも入力されずに⏎キーが押されると、変数Kotae$には空の文字が格納されます。空文字でなければ、配列D$( )に入力されたデータを代入します。

159行　入力データをフロッピーディスクに書き込みます。

（データ読み込みSUBプロシージャ）

166〜179行　フロッピーディスクに記録されたランダムファイルから、変数にデータを読み出します。

167〜170行　ファイルをオープンし、引数で渡されたレコード番号

```
328:     PRINT Menu$(MenuBangou); "  No.1 ～"; "  No."; SaidaiBangou
329:     PRINT
330:     INPUT "何番のデータを修正しますか "; Kari$
331:     Bangou = VAL(Kari$)
332:     IF Bangou < 1 OR SaidaiBangou < Bangou THEN RETURN
333:     DATA.TEISEI DATA$(), Bangou, KoumokuSuu
334:     RecordNumber = Bangou
```

```
335:     DATA.KAKIKOMI DATA$(), RecordNumber
336: END SUB
337:
338: SUB TOUROKU
339:     SaidaiBangou = SaidaiBangou + 1
340:     Bangou = SaidaiBangou
341:     DO
342:         CLS
343:         LOCATE 1, 1
344:         PRINT Menu$(MenuBangou); "  No."; Bangou
345:         DATA.HYOUJI DATA$(), KoumokuSuu
346:         FOR i = 1 TO KoumokuSuu
347:             LOCATE i * 2 + 1, 8
348:             INPUT "", Kotae$
349:             DATA$(i) = Kotae$
350:             IF DATA$(1) = "" THEN
351:                 SaidaiBangou = SaidaiBangou - 1
352:                 Bangou = SaidaiBangou
353:                 EXIT SUB
354:             END IF
355:         NEXT
356:         LOCATE KoumokuSuu * 2 + 3, 1
357:         PRINT "訂正がありますか ( 'Y'es / 'N'o )"
358:         Teisei$ = LCASE$(INPUT$(1))
359:         IF Teisei$ = "y" OR Teisei$ = "ﾝ" THEN
360:             RecordNumber = Bangou
361:             DATA.KAKIKOMI DATA$(), RecordNumber
362:             DATA.TEISEI DATA$(), Bangou, KoumokuSuu
363:         END IF
364:     LOOP WHILE Teisei$ = "y" OR Teisei$ = "ﾝ"
365:     RecordNumber = Bangou
366:     DATA.KAKIKOMI DATA$(), RecordNumber
367: END SUB
```

をセットして、ランダムファイルから1レコードのデータを変数に読み込みます。

終了するとファイルをクローズします。

171〜178行　読み込まれてユーザー定義型変数に格納されたデータを、呼び出したプログラムに引数として返します。

（印刷 SUB プロシージャ）

181〜214行　データの範囲を指定して印刷します。

186〜190行　印刷を開始する番号と終了する番号を入力します。異常な値が入力された場合、188行、190行の EXIT SUB で SUB プロシージャを脱出してメインプログラムに戻ります。

199〜212行　DO〜LOOP ステートメントで、レコード番号を更新しながら、ファイルからデータを読み出しては、プリンタで印刷しています。

（一覧 SUB プロシージャ）

216〜243行　登録されているデータを順番に表示する一覧のプロシージャです。

（検索サブルーチン SUB プロシージャ）

245〜273行　入力されたデータを検索するサブルーチンです。

251〜268行　二重の FOR〜NEXT ループです。外側のループで、全データをランダムファイルから読み出しては、順番に探していきます。内側のループで、読み込まれた配列の各要素のデータと検索文字列を比較しています。

（削除 SUB プロシージャ）

275～303行　不要になったデータのレコードに、「ヌル」（空の文字列）を書き込んで削除するサブルーチンです。

288～289行　削除するデータの番号に対応するレコード番号のデータを読み込んで、画面に表示します。

296～302行　Yキーが押されると、選ばれた番号のレコード番号のデータに、すべて空文字を代入したデータを書き込んで、データを消してしまいます。

（選択 FUNCTION プロシージャ）

305～319行　しごとのメニューを表示して、数字キーで選択します。

318行　入力された回答を、VAL関数を使って数値にし、プロシージャ名のSENTAKUに代入して、呼び出したプログラムに結果として値を返します。

（終了 SUB プロシージャ）

321～323行　変数Owariに Haiを代入してフラグを立てて、メインプログラムに終了を知らせます。

（修正 SUB プロシージャ）

325～336行　修正するデータの番号を入力して、データ訂正のプロシージャを呼び出すプロシージャです。

333行　データ訂正SUBプロシージャに引数を与えて呼び出し、訂正します。

334～335行　修正の終わったデータは、データ書き込みのSUBプロシージャを呼び出して、ファイルに登録します。

（登録 SUB プロシージャ）

338～367行　データを入力するプロシージャです。

339～340行　データの最大番号をカウントアップして、これから入力するデータの番号を最大番号にしています。

341～364行　データ入力のためにDO～LOOPを使っています。

342～355行　各項目のデータを入力する部分です。

356～363行　入力したデータに訂正がないか確認します。

365～366行　データの番号をレコード番号にして、データ書き込みSUBプロシージャを呼び出して、データを記録します。

Quick BASICテクニカルマニュアル

著　者　伊東ひろみ
発行者　富　永　弘　一
印刷所　湯島印刷株式会社

発行所　東京都台東区台東4丁目7　株式会社　新星出版社
郵便番号110　電話(831)0743　振替東京 4-72233

Printed in Japan
ISBN4-405-06095-9

新星出版社の

わかりやすい実用パソコンBOOKS

一太郎Ver.3テクニカルマニュアル
日下部孝一著 ● 1230円
機能充実のワープロソフト「一太郎 Ver. 3」をやさしく解説し、はじめて接する人でもすぐに理解できるようにした一番わかりやすい入門書。

一太郎Ver.4テクニカルマニュアル
岡田慎一著 ● 1300円
一太郎の基本操作（文章入力、漢字変換、訂正など）と、さらにレベルアップした多くの機能について、初心者でもすぐに理解できるよう解説。

花子テクニカルマニュアル
新家弘健著 ● 1230円
図形プロセッサ「花子」によるスムーズな図形処理の方法を、基本から応用まで順を追って紹介した。実例も豊富に収録。

新松テクニカルマニュアル
服部佳代著 ● 1230円
「新松」の多彩な機能を活用するために、企画書やチラシ広告など実際の文書作成を通して、その操作方法を具体的に解説した入門書。

Lotus1-2-3 テクニカルマニュアル
牧田醇一著 ● 1600円
表計算の代表的ソフトの使い方を、売上日計表、請求書発行、グラフ化などの日常業務に即して、わかりやすく解説した、実践的な入門書。

クイックベーシックテクニカルマニュアル
伊東ひろみ著 ● 1600円
統合開発環境を備えた新世代の構造化言語クイックベーシックのオペレーションとプログラミングについてわかりやすく解説した。

絵でわかる はじめてのPC-98
田中一郎著 ● 1000円
16ビットパソコンの基本的な構造とキー操作、ベーシックの使い方などを図とイラストで解説した、ビギナーのためのPC-9801入門。

早わかり MS-DOS Ver.3.3実用マニュアル
田中一郎著 ● 1000円
MS-DOSの基礎知識はもちろん、作業を効率化するバッチファイルや日本語処理の方法等を解説。全コマンドを実行例でくわしく説明。

はじめてのMS-DOS
田中一郎／大橋　均著 ● 1540円
パソコンの機能を最大限に活用したい人のために、MS-DOSの基礎知識やコマンドの使い方をわかりやすく解説した入門書。

やさしい MS-DOSの基礎知識
FIELDNUT PLANNING著 ● 1500円
MS-DOSを初めて使う人にも基本操作のマスターから、コマンド、階層化ディレクトリなど、実行例をあげていちばんやさしく解説した。

はじめての パソコン通信
小久保一男著 ● 1230円
初めてパソコン通信に取り組む人のために、その基礎知識からBBS・電子メール・チャットなどのアクセスの実際までわかりやすく紹介。

わかる パソコン通信アクセスガイド
臼井公孝著 ● 1230円
日本のPC-VANや日経テレコン等に実際にアクセスし、新聞・雑誌の検索や最新ニュースの先取りなど、活用方法を実例で紹介。

初歩のパソコン入門
三木　守著 ● 970円
パソコン操作の基礎知識とBASICによるプログラミング方法の解説を通して、コンピュータ的考え方とパソコンの初歩が理解できる本。

早わかり ベーシック用語辞典
田中一郎／小山郁夫著 ● 1000円
プログラミングに必要なベーシックのコマンドやステートメントが簡単なプログラムの作り方を通じて学べる。50音順で事典としても便利。

早わかり ベーシック決まり文句
大橋　均／田中一郎著 ● 1640円
〈自由に線を引く〉〈キャラクタを移動させる〉など、プログラミングの基本となるベーシックのステートメントを、具体的な例で説明した。

Z80 わかる機械語入門
若松登志樹著 ● 1230円
パソコンの機能をさらに深く追求したい人のために、Z80のマシン語命令を解説し、マシン語による簡単なプログラミングの仕方を解説した。

Quick BASIC

Quick BASIC

テクニカルマニュアル

オペレーションと構造化プログラミングの基本

■

新星出版社

■

定価1600円
(本体1553円)

ISBN4-405-06095-9 C2055 P1600E